Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Diseño de un sistema de detección de eventos sonoros

Autor: Julián García Miranda

Tutora: María Auxiliadora Sarmiento Vega

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Diseño de un sistema de detección de eventos sonoros

Autor:
Julián García Miranda

Tutora:
María Auxiliadora Sarmiento Vega
Profesor Titular

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado: Diseño de un sistema de detección de eventos sonoros

Autor: Julián García Miranda
Tutora: María Auxiliadora Sarmiento Vega

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia y amigos, que pese a que el camino ha sido más largo y difícil de lo esperado me han apoyado de forma incondicional.
A mi madre, cuya experiencia y fortaleza han sido mi mejor inspiración.

Julián García Miranda
Sevilla, 2024

Resumen

En este trabajo analizaremos el uso de embeddings de audio como características de entrada para el entrenamiento de un sistema de detección de eventos sonoros (SED) usando machine learning. Explicaremos el estado del arte de los sistemas SED, utilizaremos las herramientas de machine learning de MATLAB para extraer las características de los audios y para realizar el entrenamiento, finalmente analizaremos los resultados obtenidos al usar una base de datos pública de etiquetado débil para el entrenamiento del sistema.

Abstract

In this work, we will analyze the use of audio embeddings as input features to train a sound event detection system (SED) using machine learning. We will explain the state of the art of SED systems, use MATLAB's machine learning tools to extract audio features and perform the training tasks. Finally, we will discuss the results obtained by using a publicly available weakly labeled dataset to train the system.

Índice Abreviado

Índice

1 Introducción

El campo de los sistemas de detección de eventos sonoros (SED, por sus siglas en inglés) ha experimentado un notable progreso en los últimos años. Estos sistemas se dieron a conocer a los usuarios comunes como herramientas para la identificación de las canciones que se reproducían en la radio o la televisión. Las más conocidas son Shazam y TrackID, muy populares al inicio de la década de los años 2010. Sin embargo, la SED tiene muchas más aplicaciones no tan conocidas en diversos ámbitos, desde el sanitario para la detección de patologías a la seguridad. En este trabajo evaluaremos el uso de embeddings de audio para el diseño de un sistema SED en el software MATLAB empleando una base de datos de anotación débil. Las bases de datos de anotación débil aportan una información limitada sobre las muestras, indican exclusivamente el tipo de evento que ocurre durante la misma, sin incluir la posición temporal del evento ni si se repite o se solapa con algún otro. Sin embargo, estas bases de datos resultan útiles cuando no se dispone de bases de datos fuertemente anotadas, ya que estas suelen ser muy costosas y laboriosas de obtener. Aprovecharemos las herramientas que nos proporciona MATLAB para procesar las muestras y desarrollar los modelos de aprendizaje automático, explorando las posibilidades y limitaciones que tendremos al utilizar una base de datos con etiquetas débiles para crear un SED capaz de detectar y reconocer eventos con utilidades prácticas.

Figura 1.1 Logotipos de Shazam y TrackID.

1.1 Estado del Arte

En este apartado definiremos qué es un sistema de detección de eventos sonoros, sus tipos y diferencias, además de analizar el estado de desarrollo en el que se encuentra en la actualidad.

1.1.1 Qué es un SED

Los sistemas de detección de eventos sonoros (SED) se encargan de detectar clases de eventos sonoros en muestras de audio. Los SED se pueden clasificar en dos tipos principales, los sistemas monofónicos, que detectan la ocurrencia única temporal de un determinado evento sonoro en una muestra de audio, y por otra parte están los sistemas polifónicos, que pueden detectar la ocurrencia de distintos eventos en la misma muestra de audio. Un sistema SED se suele diseñar utilizando el método del aprendizaje supervisado, en el que mediante el uso de características extraídas de muestras de audio correspondiente a los eventos se entrena un clasificador, que será el que indique si ha ocurrido algún evento en la muestra analizada. Para realizar el entrenamiento, necesitamos una fuente de datos que contenga los archivos de sonido y los eventos a los que corresponden, esto es la anotación de la base de datos, que puede ser débilmente anotada o fuertemente anotada. Las bases de datos débilmente anotadas sólo indican si ocurre el evento o no, mientras que las fuertemente anotadas indican, además, la posición temporal del evento y la duración del mismo. En el caso de emplear una base de datos débilmente anotada dará lugar a un sistema SED débilmente anotado, que será útil para detectar si el evento ocurre o no en una muestra de sonido, mientras que al usar una base de datos fuertemente anotada, el resultado será un sistema SED fuertemente anotado que indicará, además de la ocurrencia o no del evento, el momento en el que se produce y la duración del mismo.

Sistemas Monofónicos

Los sistemas de detección de eventos sonoros suponen un problema complejo en su implementación debido a la diversidad de situaciones a las que se deben enfrentar. No se puede considerar un entorno real el uso de muestras de sonido aisladas que se suelen utilizar para el entrenamiento; ya que cuando el sistema esté en producción, será expuesto a ruido ambiente, así como a la superposición de otros sonidos que afectarán de manera significativa el resultado de la clasificación.

Cuando se trata la detección de eventos monofónicos como se representa en la figura 1.2, se suele interpretar como un problema de clasificación multi-clase, donde cada clase se asocia a la presencia de un evento sonoro o a la ausencia de ellos. La extracción de características del sonido se puede hacer con los coeficientes cepstrales en la escala de mel (MFCC, Mel-Frequency Cepstral Coefficients) y se pueden clasificar mediante modelos ocultos de Markov (HMM, Hidden Markov Models) o modelos de mezclas gaussianas (GMM, Gaussian Mixture Models), tal y como se expone en [1], utilizando una base de datos de sonidos aislados para entrenar y grabaciones de la vida real para realizar experimentos. En [2], se utiliza un método basado en coeficientes MFCC y GTCC (Gammatone Cepstral Coefficients) desglosado en la figura 1.3, que resulta factible para su uso en tiempo real debido a su velocidad y el clasificador que se utiliza el de máxima verosimilitud ya que todas las clases son equiprobables, el método utilizado es el de Bag-of-Super-Features como se puede observar en la figura 1.4.

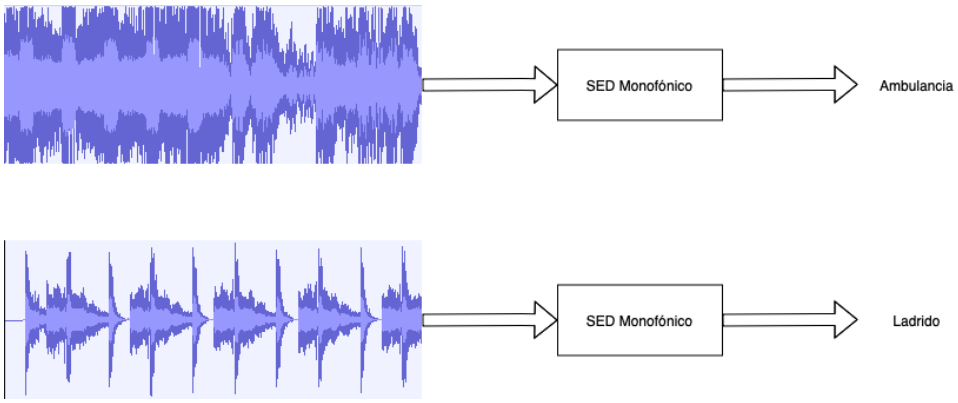

Figura 1.2 Sistema de detección de eventos monofónico..

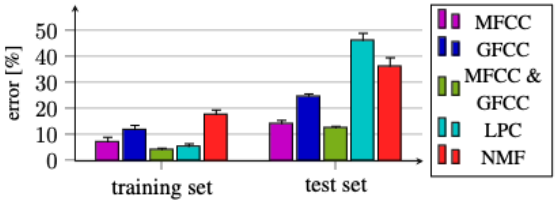

Figura 1.3 Resultados de evaluación de distintos tipos de características en [2].

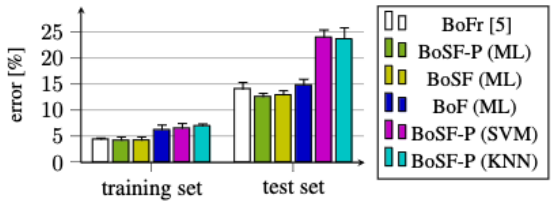

Figura 1.4 Resultados de evaluación de clasificadores en [2].

En [3] utilizan el método del Bag of Words, que consiste en el uso de *aural words*, que son las características espectrales, temporales y de energía de fragmentos de audio de corta duración, siendo el caso descrito de 32ms cada fragmento. En este método se considera que un evento sonoro ha ocurrido cuando un conjunto de palabras que representan ese evento se dan en un momento dado. En el caso descrito en el artículo lo utilizan para crear un SED con ruido de fondo empleando una clase para el ruido además de las clases de los eventos objetivo. La precisión obtenida en el artículo es de un 86.7 % utilizando un clasificador SVM lineal.

Sistemas Polifónicos

En los últimos años, se ha comenzado a desarrollar sistemas de clasificación polifónicos con el objetivo de imitar la capacidad auditiva del oído humano, utilizando distintos medios basados en redes neuronales y en métodos alternativos [4], e incluso acuñando el término "separación universal de sonidos" que es el proceso de separación de las fuentes de sonido, dando lugar a distintas pistas de audio que se utilizan en las etapas iniciales de diseño de los SED, el objetivo de los SED polifónicos es indicar el momento en el que ocurre y la duración del mismo dentro de la muestra de sonido analizada. A continuación, se aborda la problemática de la superposición de eventos sonoros, que al igual que en los sistemas monofónicos, puede interpretarse como un problema de clasificación multi-etiqueta, como se describe en la figura 1.5. Una solución común implica el uso de diccionarios pre-entrenados. En [5], se propone un sistema que extiende el análisis de componentes latentes probabilísticos (PLCA) modelado a partir de un diccionario. PLCA puede considerarse como una extensión probabilística de la fatorización de matrices no negativas (NMF). Adicionalmente, sugieren la integración de sistemas dinámicos lineales (LDS) junto con PLCA para conseguir la identificación polifónica con la información temporal de los eventos detectados, además consigue un aumento de la detección de eventos reales del 15,8 % frente al uso de PLCA únicamente. Sin embargo, en [6], se propone un modelo que utiliza NMF para separar el ruido, empleando técnicas de aprendizaje en tiempo real que analizan las características y actualizan el modelo, consiguiendo una mejora en la precisión de detección de los eventos al reducir las falsas alarmas. En [7], se propone el uso de filtrado de ruido mediante post procesado a la salida de redes neuronales profundas (DNN) de alimentación hacia adelante con múltiples etiquetas, las cuales han demostrado previamente que pueden superar el rendimiento de los métodos basados en modelos ocultos de Markov (HMM) con aumentos de precisión con respecto al original sin post procesado del 19 %. Mientras que en [8], se propone el uso de CNN para la clasificación de sonidos ambientales, obteniendo como resultado que el modelo mejora el rendimiento de las implementaciones basadas en MFCC, consiguiendo un aumento de precisión del 10.1 % con la base de datos ESC-50 y de un 0.6 % con UrbanSound8K. En [9], se presenta el uso de redes neuronales recurrentes bidireccionales de memoria a corto y largo plazo (BLSTM-RNN), obteniendo resultados que superan ampliamente los modelos de clasificación convencionales con un 15 % de mejora de media por frame y de un 6.8 % para bloques no solapantes de 1 segundo. Por otro lado, en [10], se realiza una combinación de redes neuronales recurrentes y convolucionales (CRNN) que también mejora los resultados de los modelos separados, aunque con algunas limitaciones.

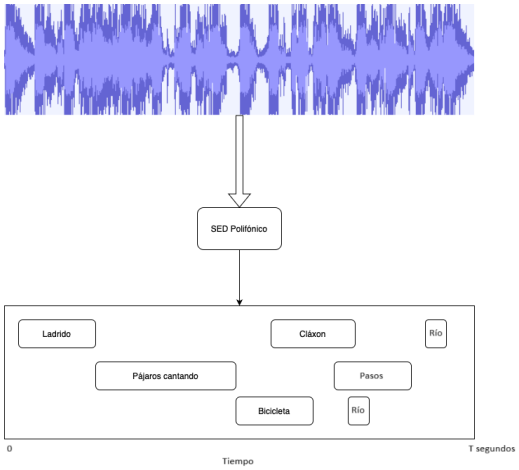

Figura 1.5 Sistema de detección de eventos polifónico.

1.1.2 Características para la detección de eventos sonoros

En este apartado explicaremos cuales son los tipos de características más utilizados en los sistemas SED y en qué consisten.

Características Espectrales

- **Centroide Espectral** El centroide espectral es una medida importante que nos indica la media de las frecuencias en una señal de audio. Es útil para ver cómo cambian las frecuencias dominantes con el tiempo, y se usa mucho en aplicaciones como el procesamiento de la voz. El centroide espectral nos indica en qué frecuencias se concentra la mayor parte de la energía y se calcula de la siguiente forma [11]:

$$SC = \frac{\int_0^\infty \overline{w}\omega \left|F(\omega)\right|^2 d\omega}{E} \tag{1.1}$$

  donde $\omega = \omega_k$ representa la frecuencia central, E representa la energía y $|F(\omega)|^2$ representa el espectro de potencia de la señal. Estas características se pueden usar para distinguir muestras de habla de las de música [11].

- **Rolloff Espectral** El rolloff espectral calcula la frecuencia F por debajo de la cual se concentra un cierto porcentaje de la energia total de la señal. El porcentaje suele rondar el 90 %. Si el m-ésimo coeficiente de la transformada discreta de Fourier (DFT) corresponde al rolloff espectral del segmento i-ésimo, este satisface la siguiente ecuación [12]:

$$\sum_{k=1}^{m} X_i(k) = C \sum_{k=1}^{W_f L} X_i(k) \tag{1.2}$$

  en la que $X_i(k)$ el valor del k-ésimo coeficiente de la DFT del segmento i-ésimo de la señal. Representa la magnitud espectral de la señal en la frecuencia correspondiente al índice k. C es el porcentaje elegido [0,85-0,99][11], la frecuencia de rolloff se suele dividir entre $W_f L$, que es el número de coeficientes de DFT usados, para que los valores sean entre 0 y 1, siendo 1 el valor de frecuencia máxima. W_f controla el tamaño ventana que se usa en la segmentación de la señal antes de aplicar la transformada discreta de Fourier (DFT). La señal se divide en segmentos, y W_f define cuántas muestras de la señal se consideran en cada segmento. L se refiere a la longitud de la DFT, es decir, el número de puntos que se calculan en la transformada discreta de Fourier para cada segmento de la señal. La frecuencia de rolloff viene dada por:

$$F = \frac{m}{W_f L} \tag{1.3}$$

- **Flujo Espectral** Las características de flujo espectral miden las diferencias espectrales entre dos segmentos sucesivos de la señal y se calcula mediante la siguiente ecuación[12]:

$$Fl_{(i,i-1)} = \sum_{k=1}^{W_f L} \left(EN_i(k) - EN_{i-1}(k)\right)^2 \tag{1.4}$$

  en la que $EN_i(k) = \frac{X_i(k)}{\sum_{l=1}^{W_{fL}} X_i(l)}$ donde $EN_i(k)$ es el coeficiente DFT k-ésimo normalizado en el segmento i-ésimo y W_{fL} es el número de coeficientes DFT.

- **Coeficientes Ceptrales en las frecuencias de Mel** Tal y como desarrollan Zrar y Abdulbasit en su artículo[13], los coeficientes ceptrales en la frecuencia de Mel (MFCC) son unos de los métodos de extracción de características de los sonidos más utilizados en el ámbito SED para realizar clasificaciones relacionadas con la voz humana. Dependiendo de su implementación, explican que pueden ser calculados de la siguiente forma :

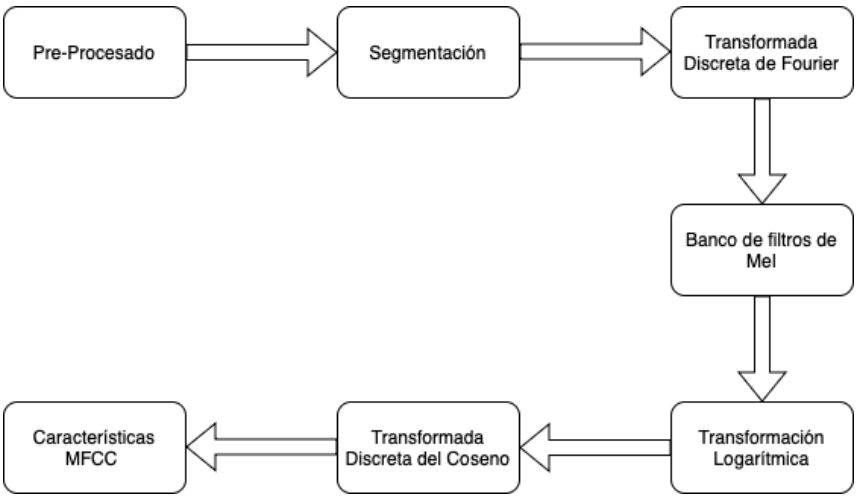

Figura 1.6 Proceso de cálculo de las características MFCC.

1. Pre-Procesado: En esta primera fase se realiza un procedimiento común de compensación de los componentes de alta frecuencia de las muestras de sonido. En este caso se puede ejecutar aplicando un filtro de paso alto con ajuste [1,-0.97].

2. Segmentación de la señal: La segmentación es un proceso que permite la separación de los datos de la muestra de sonido en frames. Estos frames tienen una duración corta, usualmente de 20ms, donde la señal tiende a ser más estacionaria, lo que facilita el análisis de la muestra, además la superposición entre frames es de 10ms. En cada frame se aplica la ventana de Hanning y Hamming, ya que mejoran armónicos, suavizan la señal y reduce el efecto edge en conjunto con el siguiente paso.

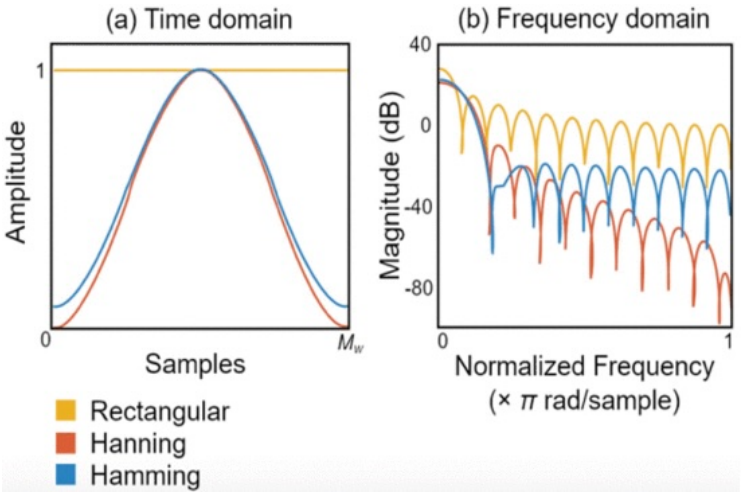

Figura 1.7 Ventanas de Hanning y Hamming rectangulares en el dominio del tiempo y la frecuencia[13].

3. Transformada Discreta de Fourier: La Transformada Discreta de Fourier (DFT) permite obtener el espectro densidad de potencia de las componentes en frecuencia que forman la señal. La transformada de cada frame viene dada por la siguiente ecuación:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi jnK}{N}} \quad k = 1,2,3 \ldots N-1 \tag{1}$$

donde *x(n)* es el frame y N es su longitud.

4. Banco de filtros Mel: La señal pasa por unos filtros, que se construyen basados en la percepción del tono, aunque fueron desarrollados originalmente para el análisis de la voz y de la percepción del sonido de las personas, el objetivo final es extraer una representación de la señal no lineal. Está constituido por cuarenta filtros triangulares con la siguiente función de transferencia:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leqslant k < f(m) \\ 1 & k = f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \leqslant f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{2}$$

donde *f(m)* es la frecuencia central del filtro y $\sum_{m=0}^{M-1} H_m(k) = 1$. La escala Mel en la frecuencia de respuesta y viceversa viene dada por estas funciones:

$$m = 2595 log_{10}\left(1 + \frac{f}{700}\right) \tag{3}$$

$$f = 700\,(m/2595) \tag{4}$$

5. Transformación Logarítmica: Se convierten las unidades naturales en unidades logarítmicas de base 10.

6. Transformada del Coseno: La Transformada Discreta del Coseno (DCT) se aplica tras aplicar la transformación logarítmica para seleccionar los coeficientes con mas variabilidad a la hora de obtener los MFCC. Esta transformada viene dada por la siguiente ecuación

$$Y(k) = \sum_{n=0}^{N-1} y_n * cos\,(2\pi jnk/N), \quad k = 1,2,3 \ldots N-1 \tag{5}$$

donde *y(n)* es el frame y N es su duración.

7. Características MFCC: Coeficientes MFCC obtenidos al finalizar el proceso que corresponden a las características de la muestra analizada.

Un espectrograma es la representación de las componentes de frecuencia de una señal, en nuestro caso de sonido. Se genera mediante un proceso similar al de obtención de los coeficientes de Mel, en el que hay que segmentar la señal en frames y aplicar la transformada discreta de Fourier. Se representan las frecuencias y su intensidad en el tiempo.

- **Coeficientes ceptrales de gammatono** En el artículo de Xavier Valero y Francesc Alias [14] se analiza la extracción de características utilizando este método que se basa en una modificación de los filtros de gammatono con inspiración biológica. En este caso se utilizan para la detección de sonidos no relacionados con el habla. El filtro gammatono recibe su nombre del impulso $g(t)$, que es el producto de una función distribución gamma y un tono senoidal centrado en f_c, siendo la ecuación:

$$g(t) = Kt^{(n-1)}e^{-2\pi Bt}\cos(2\pi f_c t + \varphi) \quad t > 0 \tag{1.5}$$

donde:
K es la amplitud,
n es el orden del filtro,
f_c es la frecuencia central en Hercios,
φ es el desfase y
B representa la duración del impulso respuesta.

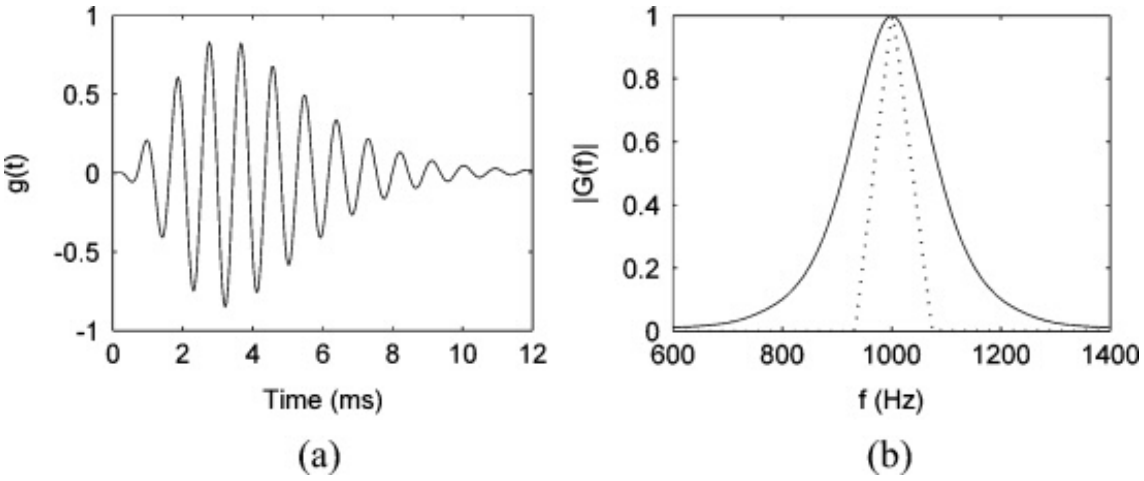

Figura 1.8 Respuesta impulsiva de un filtro gammatono centrado a 990Hz (a) junto a la respuesta en frecuencia del mismo filtro superpuesto a un filtro Mel (b) siendo la línea continua el filtro gammatono y el filtro de Mel la línea discontinua[14].

El ancho de banda del equivalente rectangular (ERB) es una medida que describe cómo nuestro oído procesa los sonidos. Representa la capacidad de la cóclea, parte interna del oído, para distinguir entre diferentes frecuencias sonoras, la longitud del filtro gammatono está relacionada con la ERB, ya que ambos conceptos tratan de reflejar cómo el sistema auditivo humano percibe las frecuencias de manera no lineal: a frecuencias bajas, los filtros son más largos y más ajustados; a frecuencias altas, son más cortos y amplios. Si modelamos el oído como una serie de filtros, cada uno captando un rango específico de frecuencia. Las células ciliadas en la cóclea detectan estos sonidos filtrados y envían la información al cerebro. El ERB cuantifica el ancho de banda de estos filtros en diferentes puntos de la cóclea, indicando la precisión con la que podemos diferenciar tonos cercanos en distintas frecuencias.

$$\text{EBR} = \left[\left(\frac{f_c}{\text{EarQ}} \right)^p + (\text{mín}(\text{BW}))^p \right]^{1/p} \tag{1.6}$$

donde $EarQ$ es la calidad de filtro asintótico a alta frecuencia, $minBW$ es el ancho de banda mínimo a baja frecuencia y p es comúnmente 1 o 2.

La distribución de los filtros es crucial al diseñar un banco que simule el oído humano. Aunque la cóclea contiene alrededor de 3000 células ciliadas, cada una sensible a una frecuencia específica, en los SED se suele simplificar. En lugar de replicar exactamente la complejidad del oído, se utiliza un número menor de filtros de paso de banda. Estos filtros se superponen parcialmente en el espectro de frecuencias, imitando así la respuesta auditiva humana de manera más simple. Basándonos en el ERB, que refleja las características biológicas del oído, las frecuencias centrales f_{ci} de cada filtro en este modelo simplificado se calculan mediante la siguiente fórmula:

$$f_{ci} = \left(f_{hight} + EarQminBW\right) e^{-istep/EarQ} - EarQminBW \tag{1.7}$$

donde:
f_{high} es la frecuencia más alta del banco de filtros, $EarQ$ y $minBW$ son los parámetros ERB, i es el índice del filtro y $step$ es el salto entre filtros consecutivos, que puede ser calculado como:

$$step = \frac{EarQ}{N} \log \left(\frac{f_{hight} + EarQminBW}{f_{low} + EarQminBW} \right) \tag{1.8}$$

donde: f_{low} es la frecuencia mínima y N es el número de filtros.

El cálculo de los coeficientes GCC sigue un proceso similar a la extracción de MFCC. La señal de audio se segmenta en fragmentos cortos, típicamente de 10 a 50 ms, los denominados frames. Esta segmentación se hace con dos objetivos, permite tratar la señal como estacionaria en un intervalo suficientemente pequeño y mejora la eficiencia en la extracción de características. Posteriormente, se aplica la transformada rápida de Fourier (FFT) a cada frame. El banco de filtros se aplica sobre esta transformada, resaltando las componentes sonoras más relevantes para la percepción auditiva humana.

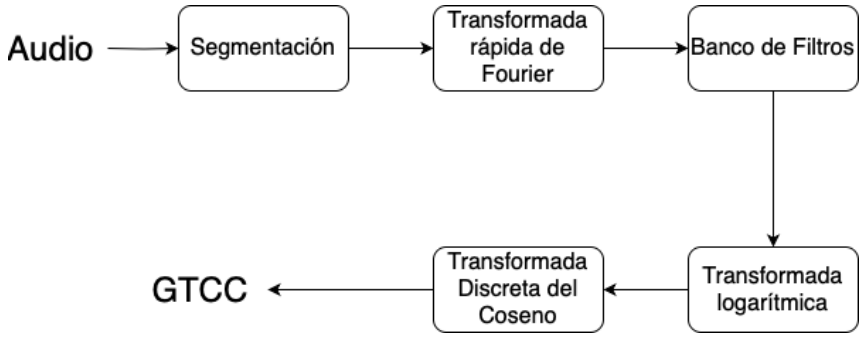

Figura 1.9 Diagrama de obención de los GTCC.

Finalmente, la función logarítmica y la transformada discreta del coseno son aplicadas para modelar la percepción humana del volumen. El coste computacional de este proceso es prácticamente igual que el del cálculo de las MFCC.

$$GTCC_m = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} \log(X_n) \cos\left[\frac{\pi n}{N}\left(m - \frac{1}{2}\right)\right] \quad 1 \leq m \leq M \tag{1.9}$$

donde:
X_n es la energía de la señal en la enésima banda espectral,
N es el número de filtros de gammatono,
M es el número de coeficientes GTCC.

Embeddings

Las redes neuronales pueden estar constituidas por un número elevado de capas conectadas entre si de diferentes formas, es necesario entrenar la red para un determinado caso de uso y una vez entrenada, se pueden usar las salidas de las capas intermedias como características relevantes que pueden ser útiles en problemas similares o relacionados de alguna forma con el entrenamiento de la red. Las salidas de estas capas intermedias son los embeddings [15], unas características muy potentes para realizar sistemas de detección de eventos sonoros. En el caso de los embeddings para audio, hay distintas redes pre entrenadas para clasificación de eventos sonoros, las más comunes son VGGish [16], Kumar [17], L^3-Net [18] y OpenL3 [19].

En el caso de VGGish, sus inicios vienen de la red VGG, que fue ideada originalmente para la clasificación de imágenes. En una de las primeras veces que se modificó la red VGG para utilizarla con muestras de audio, se realizó el entrenamiento de una forma supervisada empleando la base de datos Million Song Dataset. Aunque estos embeddings resultaron ser más eficaces que los coedicientes MFCC, seguían por debajo del rendimiento del estado del arte excepto para la clasificación del habla y música [20]. Los embeddings VGGish fueron propuestos en [16] por primera vez, en ese artículo utilizaron la base de datos YouTube-100M, que es de etiquetado débil, por lo que las etiquetas no indican el momento temporal en el que ocurren los eventos y entrenaron diferentes redes con el objetivo de evaluar su rendimiento en la clasificación de eventos sonoros. Además, realizaron modificaciones en la última capa de la red VGG y aplicaron el uso de normalización por lotes en vez de normalización de respuesta local, de aquí surgió VGGish.

Los embeddings de Kumar [17] fueron creados entrenando una red CNN profunda utilizando la base de datos de etiquetado débil Audioset, tiene etiquetas correspondientes a 527 eventos sonoros. En el artículo [17] realizaron pruebas con los embeddings en un SED con la base de datos ESC-50 donde alcanzaron la precisión del 83,5 %, siendo esta mayor que la precisión alcanzada por humanos con esta base de datos.

Los embeddings L^3-Net [18] fueron creados al entrenar la red con vídeos sin etiquetar aplicando correspondencia audiovisual, es decir, un evento visual corresponde se con un evento sonoro. Se utiliza un video con audio sincronizado y otro de-sincronizado en dos ramas separadas de CNN, una para la extracción de características de vídeo y otra para las de audio. Las salidas de ambas redes se combinan, y se entrena un clasificador para detectar si el audio corresponde o no al vídeo. De esta forma, grandes cantidades de vídeos sin etiquetar pueden ser utilizadas para entrenar la red de manera auto-supervisada. Este método se llama fusión tardía.

Los embeddings de OpenL3 [19] son una extensión de L^3-Net, en los que se utiliza AudioSet para crear distintos modelos entrenados de forma auto-supervisada, igual que L^3-Net. En el artículo [19] evalúan las consecuencias que tendría alterar algunos parámetros de los embeddings, como podía ser su tamaño o tipo de sonidos usados en el entrenamiento.

El tipo de sonidos usados en el entrenamiento no tiene por qué ser el mismo que el de la tarea final de clasificación y en el artículo [19] han demostrado que realizar el entrenamiento con el mismo tipo de sonidos que tiene que clasificar el SED final no es beneficioso, siendo el uso de una cantidad de muestras de audio suficiente para el entrenamiento lo más importante para conseguir unos embeddings adecuados. Para entrenar la red han utilizado la base de datos Audioset, descargando 2 millones de vídeos con sus correspondientes pistas de audio, y dividiéndolos en dos tipos: sonidos ambientales y música. OpenL3 ha demostrado ser superior a VGGish en las pruebas realizadas con varias bases de datos ESC[20].

Características Temporales

- **Tasa de los cruces por cero** La tasa de cruces por cero indica el número de veces que la señal pasa por cero en un periodo, lo que equivale al número de cambios de signo del segmento en el caso de que se analice al completo. Estas características pueden indicar cual es el comportamiento espectral por lo que se puede decir que la tasa de cruces por cero sirve como aproximación de la naturaleza espectral de la señal [11]. La tasa de cruces por cero viene definida por la siguiente ecuación:

$$ZC = \frac{1}{2N}\sum_{n=1}^{N} |\text{sgn}[x(n)] - \text{sgn}[x(n-1)])| \tag{1.10}$$

en la que $sgn[x(n)]$ es la función:

$$\text{sgn}(x_i(n)) = \begin{cases} 1, & x_i > 0, \\ -1, & x_i < 0. \end{cases} \tag{1.11}$$

- **Energía a corto plazo** Las características de energía a corto plazo utilizan la energía para extraer información de la señal[11]. La señal de audio se divide en segmentos solapados y las características se calculan para cada segmento de forma individual mediante la siguiente ecuación[12]:

$$E(i) = \frac{1}{W_L}\sum_{n=1}^{W_L} |x_i(n)| \tag{1.12}$$

en la que $x_i(n)$ con $n = 1,...,W_L$ es la secuencia de de muestras de audio en el segmento i-ésimo de duración W_L.

Características Prosódicas

- **Frecuencia Fundamental o Tono** La frecuencia fundamental o tono de una señal es una de las características prosódicas de mayor importancia en señales de audio. Las carácterísticas prosódicas hacen referencia a la entonación, el ritmo y el acento en el habla. Las características de frecuencia fundamental se calculan usando períodos más largos uniendo varios segmentos en conjunto con otras características del audio, como las del espectro. En lugar de usar el valor exacto del tono, se puede calcular el promedio, el valor máximo y la desviación estándar del tono en toda la señal de audio[11].
- **Intensidad o Volumen** Estas características se extraen por segmento y representan la energía de las señales de audio. Estas se basan en el modelado de la percepción humana del volumen, en la que la respuesta auditiva crece de forma logarítmica y en la que la comprensión de los eventos sonoros también dependen de la distribución espectral de la señal y su duración [11].
- **Ritmo o Duración** Estas características se centran en la duración y el ritmo de la muestra analizada centrándose en dos propiedades de la muestra [11], la primera es la variación del ritmo a lo largo de la misma y la segunda se refiere a la duración de los fonemas, las sílabas o las pausas. Al igual que en las características de frecuencia fundamental o tono se pueden aplicar métodos de normalización.

1.1.3 Clasificadores para la detección de eventos sonoros

Modelos Ocultos de Markov (HMM)

Los HMM son máquinas de estado finito en los que los estados como bien dice el nombre no son visibles, estos estados varían cada unidad de tiempo siguiendo un proceso que varía según el modelo. En THK Book [21] analizan el uso de los HMM para la separación de de eventos sonoros ligados al procesamiento del habla en un SED. Un modelo oculto de markov discreto está definido por los siguientes parámetros:

1. $\pi_i = P(q_1 = i)$: probabilidad de que el primer estado q_1 sea i.
2. $a_{ij} = P(q_{t+1} = j \mid q_t = i)$: probabilidad de transición de un estado desde el estado actual i al estado j.
3. $b_j(k) = P(o_t = k \mid q_t = j)$, probabilidad del símbolo de observación k dado el estado actual j.

En la figura se muestra un ejemplo de este proceso donde la máquina de estados, con 6 posibles estados se mueve por la secuencia $X = 1,2,2,3,4,4,5,6$ para generar la secuencia o_1 a o_6 que son vectores de palabras observados en el tiempo t.

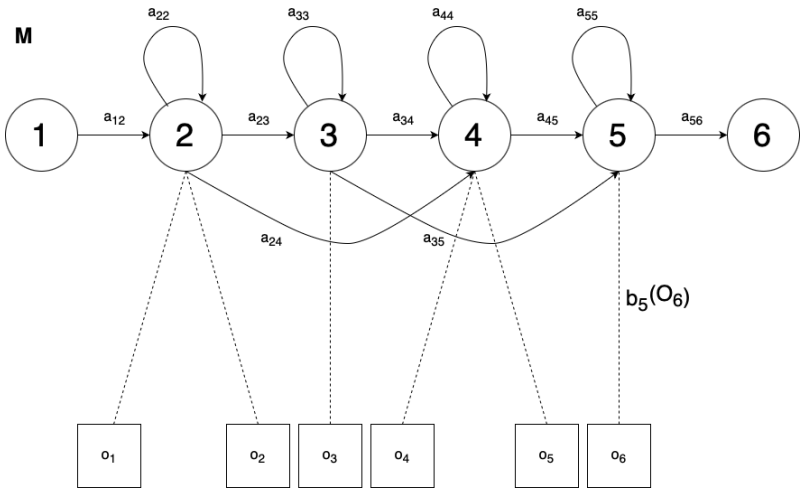

Figura 1.10 Modelo de generacion de Markov.

El modelo M puede generar la secuencia de observaciones O siguiendo una serie de estados X. Para calcular la probabilidad total, multiplicamos dos tipos de probabilidades, la probabilidad de pasar de un estado a otro (probabilidades de transición) y la probabilidad de que cada estado produzca la observación correspondiente (probabilidades de salida). Una vez hecho este proceso podremos obtener la probabilidad total de que el modelo M genere la secuencia O pasando por los estados X.

$$P(O,X \mid M) = a_{12}b_2(o_1)a_{22}b_2(o_2)a_{23}b_3(o_3)\ldots \tag{1.13}$$

Aún así, en la práctica sólo la secuencia de observación O es conocida, a diferencia de la secuencia de estados X, de ahí viene el nombre Modelo Oculto de Markov. Dado que X es desconocido, la probabilidad requerida se calcula sumando todas las secuencias posibles $X = x(1),x(2),x(3),\ldots,x(T)$ de forma que:

$$P(O \mid M) = \sum_X a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t) a_{x(t)x(t+1)} \tag{1.14}$$

donde $x(0)$ es el estado de entrada del modelo y $x(T+1)$ es el estado de salida.

En lugar de considerar todas las posibles secuencias de estados, podemos simplificar el cálculo utilizando únicamente las secuencia de estados que tienen la mayor probabilidad de ocurrir. Esto nos lleva a una nueva forma de calcular la probabilidad, que se puede expresar con una ecuación modificada:

$$\hat{P}(O \mid M) = \max_{X} \left\{ a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t) a_{x(t)x(t+1)} \right\} \tag{1.15}$$

Los HMM destacan por su sofisticación y eficacia debido a que cuando se dispone de un conjunto adecuado de datos de entrenamiento para un modelo específico, es posible ajustar sus parámetros mediante un método de recalculo. Además, si se cuenta con una cantidad suficiente de muestras para cada palabra, se puede desarrollar un modelo HMM que capture las variaciones del habla que se encuentran en situaciones reales [21]. En el ámbito de un SED, los estados ocultos de los HMM equivalen a las clases de los eventos a clasificar, mientras que las observaciones se corresponden con las características de la muestra de audio.

Modelos de mezcla Gaussiana

Tal y como expone Douglas Reynolds [22] un modelo de mezcla gaussiana (GMM, por sus siglas en inglés) es una función de densidad de probabilidad que se construye combinando varias distribuciones gaussianas. Estos modelos son muy utilizados en la estadística y el machine learning, especialmente cuando se trata de representar distribuciones de datos continuos. Un campo donde destacan en especial es en los sistemas biométricos, como el reconocimiento de voz. Aquí, los GMM se utilizan para modelar las características espectrales del habla humana. Matemáticamente, un GMM se expresa como la suma ponderada de M componentes gaussianas. Aunque la ecuación puede parecer compleja a priori, la idea es sencilla, pues se realiza una combinación de varias distribuciones normales, cada una con su propio peso, para crear una distribución más compleja que se ajuste mejor a los datos. Un modelo GMM es una suma ponderada de M funciones gaussianas dado por la ecuación:

$$p(x \mid \lambda) = \sum_{i=1}^{M} w_i g(x \mid \mu_i, \Sigma_i) \tag{1.16}$$

donde x es un vector de datos D-dimensional continuo,
w_i con $i = 1,...,M$ es la ponderación de cada elemento y
$g(x|\mu_i, \Sigma_i)$ con $i = 1,...,M$, son las componentes gaussianas a ponderar.
Cada componente es una funcion gaussiana D-dimensional dada por la función:

$$g(x \mid \mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i) \right) \tag{1.17}$$

con el vector media μ_i y la matriz covarianza Σ_i. El sumatorio de las ponderaciones satisface $\sum_{i=1}^{M} w_i = 1$. Los modelos GMM completos son parametrizados por vectores media, matrices covarianza, y ponderaciones de todos los componentes gaussianos. Estos parámetros se representan con la siguiente notación:

$$\lambda = \{w_i, \mu_i, \Sigma_i\} \quad i = 1,...,M \tag{1.18}$$

Esto permite elegir el tipo de GMM que mejor se ajuste a nuestras necesidades, ya que los datos con los que se trabajan pueden variar bastante dependiendo del uso. En un SED el modelo GMM se entrena usualmente mediante métodos iterativos EM en los que aprende las clases objetivo, cada gaussiana puede representar un evento sonoro en un SED multiclase, mientras que en uno binario varias gaussianas pueden representar una sola clase. Una vez entrenado, el modelo GMM clasifica los eventos calculando las probabilidades posteriores de las variables latentes dada la observación. La variable con probabilidad superior se suele asignar como generadora de la observación.

En [23] utilizan el modelo GMM para crear un SED que detecte cuando una persona ha tenido una caída en una vivienda, separando los eventos sonoros de las caídas del ruido de fondo. Para ello hay distintas gaussianas que modelan cada clase de evento sonoro, una para los eventos de caída y otras dos para distintos tipos de ruido.

Sistemas dinamicos lineales

Para modelar información que se desarrolla en el tiempo, podemos usar estructuras llamadas cadenas de Markov con variables ocultas. En estos modelos, lo que observamos en cada momento depende de un estado invisible o "latente". Cuando estos estados ocultos solo pueden tomar un número limitado de valores, tenemos lo que se conoce como Modelo Oculto de Markov (HMM). Si permitimos que estos estados puedan variar de forma continua, obtenemos los Modelos de Estado Espacial (SSM). Un tipo especial de SSM es el Sistema Dinámico Lineal (LDS). En este, tanto los estados ocultos como lo que observamos siguen patrones de distribución gaussiana, y cada estado se relaciona linealmente con el anterior. En el tratamiento de señales, los LDS son más conocidos como filtros de Kalman. Aunque los filtros de Kalman y los HMM se crearon por separado, últimamente se ha empezado a ver más claramente cómo se relacionan, especialmente en el campo del aprendizaje automático.

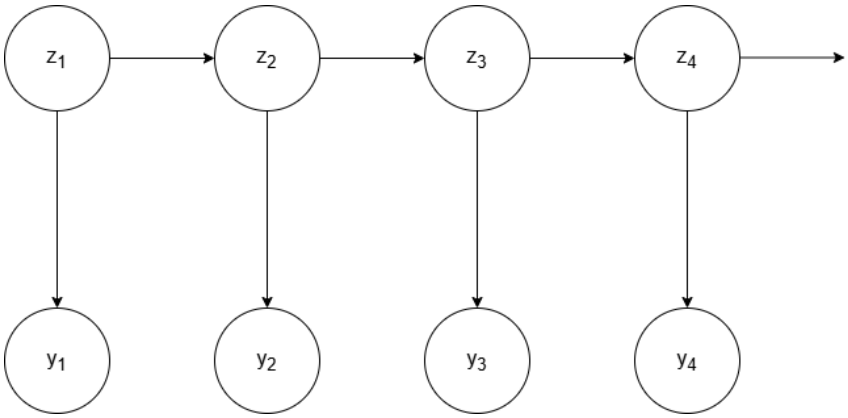

Figura 1.11 Representación gráfica de un LDS.

La representación anterior es equivalente a la de un HMM con la excepción de que en un HMM la variable latente z_t es discreta y unidimensional. Un LDS se puede formular de la siguiente forma:

$$z_t = A_t z_{t-1} + \varepsilon_t \tag{1.19}$$
$$y_t = B_t z_t + \delta_t \tag{1.20}$$
$$\varepsilon_t \sim \mathcal{N}(0, Q_t) \tag{1.21}$$
$$\delta_t \sim \mathcal{N}(0, R_t) \tag{1.22}$$

donde:
z_t es el estado oculto,
A_t es el modelo de transición,
ε_t es el ruido gaussiano del sistema con covarianza Q_t,
y_t es la observación,
B_t es el modelo de observación,
δ_t es el ruido gaussiano de la observación con covarianza R_t.

Para la clasificación de eventos sonoros, se entrena un LDS para cada clase de evento dando así un conjunto de observaciones y estados ocultos para cada clase, posteriormente para asignarle la clase se comparan las observaciones y estados ocultos de la muestra a analizar y se comparan con los valores de los modelos de entrenamiento para asignarle una clase. En [5] utilizan LDS para realizar seguimiento de los eventos sonoros en SED polifónicos, consiguiendo detectar múltiples eventos sonoros de forma concurrente.

Factorización de matrices no negativas

La factorización de matrices no negativas (NMF) es un método que descompone una matriz positiva en dos matrices positivas. Partiendo de una matriz con valores positivos $\mathbf{V}\ FxT$ que suele representar la magnitud del espectrograma de una señal, donde F es el número de frecuencias y T el número de segmentos temporales, la NMF intenta crear dos matrices más simples, $\mathbf{W}$ y $\mathbf{H}$, que al multiplicarse se parezcan lo más posible a $\mathbf{V}$. Todos los valores en $\mathbf{W}$ y $\mathbf{H}$ deben ser positivos o cero. En este caso, $\mathbf{W}$ sería como un diccionario de "sonidos básicos", y $\mathbf{H}$ indicaría cuándo y con qué fuerza aparece cada uno de estos sonidos. Para obtener las matrices W y H se plantea el siguiente problema de optimización:

$$\min_{\mathbf{W},\mathbf{H}} \mathbf{D}(\mathbf{V} \mid \mathbf{WH}) = \sum_{f,t} d(\mathbf{V}(f,t) \mid [\mathbf{WH}]_{f,t}) \tag{1.23}$$

$$\text{s.a.} \quad \mathbf{W}(f,r) \geq 0, \quad \mathbf{H}(r,t) \geq 0$$

donde:
$D(\cdot \mid \cdot)$ denota el coste o la función distancia y $d(\cdot \mid \cdot)$ es una función escalar del coste. En [24] utilizan la divergencia generalizada de Kullback-Leibler (KL) como medida de distancia, que se define como:

$$d(x \mid y) = x \log\left(\frac{x}{y}\right) - x + y \quad \text{for } x, y \in \mathbb{R}^+$$

Existen diferentes reglas de actualización de las matrices indicadas en [24], en el artículo siguen la regla de actualización multiplicativa y vienen dadas por:

$$\mathbf{W} \leftarrow \mathbf{W} \odot \left(\frac{\mathbf{V}}{\mathbf{WH}}\mathbf{H}^T\right) / (\mathbf{1}\mathbf{H}^T) \tag{1.24}$$

$$\mathbf{H} \leftarrow \mathbf{H} \odot \left(\mathbf{W}^T \frac{\mathbf{V}}{\mathbf{WH}}\right) / (\mathbf{W}^T\mathbf{1}) \tag{1.25}$$

Teniendo las matrices $\mathbf{W}$ y $\mathbf{H}$ inicializadas con valores aleatorios no negativos, las reglas de actualización mantendrán la restricción de que los valores tienen que ser iguales o mayores que 0 en las iteraciones. Considerando el problema de detectar eventos sonoros en un SED, en el artículo[24] proponen el uso del espectrograma de una entrada ruidosa $\mathbf{V}$, teniendo $\mathbf{V} \approx \mathbf{V}_s + \mathbf{V}_n$ usando s para indicar los eventos objetivo y n para el ruido. Suponiendo que la información de las dos clases de sonido está disponible como caso supervisado, un diccionario de eventos y un diccionario de ruido pueden ser entrenados previamente mediante un NMF estándar, notado por $\mathbf{W}_s \in \mathbb{R}_+^{F \times R_s}$ y $\mathbf{W}_n \in \mathbb{R}_+^{F \times R_n}$, donde R_c es el número de bases para cada fuente de sonido $c = s$ o n. El modelo NMF de la matriz V para un NMF toma la siguiente forma:

$$\mathbf{V} \approx \mathbf{WH} = \begin{bmatrix}\mathbf{W}_s & \mathbf{W}_n\end{bmatrix} \begin{bmatrix}\mathbf{H}_s \\ \mathbf{H}_n\end{bmatrix} \tag{1.26}$$

De forma similar la fórmula de actualización vienen dada por:

$$\mathbf{H}_c \leftarrow \mathbf{H}_c \odot \left(\mathbf{W}_c^T \frac{\mathbf{V}}{\mathbf{WH}} \right) / \left(\mathbf{W}_c^T \mathbf{1} \right) \quad c = s,n. \tag{1.27}$$

En el caso de que sea semi supervisado y no esté disponible el diccionario de eventos o ruido, la correspondiente matriz diccionario puede ser aprendida mediante la siguiente regla:

$$\mathbf{W}_c \leftarrow \mathbf{W}_c \odot \left(\frac{\mathbf{V}}{\mathbf{WH}} \mathbf{H}_c^T \right) / \left(\mathbf{1} \mathbf{H}_c^T \right) \quad c = s,n. \tag{1.28}$$

Clasificadores KNN (K-Nearest-Neighbor)

Este clasificador, pese a ser sencillo, es una buena opción tanto para problemas de clasificación binaria o multiclase[12]. Su punto fuerte reside en que no necesita estrictamente una fase previa de entrenamiento, pues puede realizarlo directamente durante la fase de clasificación. Su funcionamiento es tal que dado un vector de características x detectamos al principio los k vecinos más cercanos en el conjunto de datos de prueba y contamos cuantos pertenecen a cada clase, después el vector será asignado a la clase cuyo número de vecinos sea mayor. En el primer paso para detectar los vecinos más cercanos hay que calcular la distancia entre x y el resto de vectores de características v_i con $i = 1,...,M$ del conjunto donde M es el número de muestras. La forma más común de cálculo de distancias es la distancia Euclídea dada por:

$$d(\mathbf{x}, \mathbf{v}_i) = \sqrt{\sum_{j=1}^{D} \left(x(j) - v_i(j) \right)^2} \tag{1.29}$$

En la que D es la dimensión del espacio vectorial de las características. Como resultado, tendremos los k vecinos más cercanos, de los que siendo k_i el número de vectores que corresponden a la clase i-ésima con $i = 1,...,N_c$ entre los vectores más cercanos, el vector analizado será asignado a la clase para la que k_i es más alta. La probabilidad a posteriori se puede calcular mediante la siguiente ecuación:

$$P(\omega_i \mid \mathbf{x}) = \frac{k_i}{k}, \quad i = 1, \ldots, N_c \tag{1.30}$$

Árboles de Decisión

Los árboles de decisión se usan ampliamente en machine learning y data mining, siendo estos de dos tipos: *classification trees* que devuelven una clase y *regression trees* que devuelven un número real[12]. En el caso de los sistemas de detección de eventos sonoros interesan los árboles de clasificación, que toman decisiones de manera secuencial cada vector de características. En los árboles de decisión los nodos externos se llaman *hojas*, y son a los que se les asignan las clases. Para llegar a las *hojas* hay que tomar unas decisiones previas dando lugar a nodos internos en los que se van separando los vectores de características, el proceso es el siguiente:

1. Cada nodo interno es un conjunto de muestras y se separan en base a un umbral de decisión, si el valor del vector de características F es nayor que T este vector se agrupa en un nodo, si es menor se dirige a otro nodo. Esto se hace vector por vector y nodo por nodo hasta que se llega al nodo final u *hoja*.
2. A cada nodo final u *hoja* se le asigna una clase
3. El vector de características se clasifica después de haber pasado por todos los nodos internos desde el nodo raíz y se le asignará la clase del nodo final u *hoja* en la que haya acabado.

Máquinas de Vectores de Soporte (SVM)

Los clasificadores SVM se utilizan, al igual que los árboles de decisión, en el ámbito del machine learning. Es un método de clasificación que se basa en observaciones clave, que en un caso binario, el hiperplano de decisión óptimo es el que maximiza la distancia entre el conjunto de datos de entrenamiento de las dos clases[12]. El hiperplano en este caso sería una línea al tratarse de un caso binario. A su vez se utilizan hiperplanos adicionales que son paralelos al de decisión para minimizar el error de clasificación. Este método de clasificación permite el uso de características sin clase asignada, pues existe el caso no separable en el que acabarán asignadas.

1.1.4 Bases de Datos

Los sistemas de detección de eventos sonoros (SED) son métodos supervisados en su mayoría, por lo que es necesario contar con bases de datos etiquetadas para entrenar el clasificador. Debido a la prevalencia del uso de los SED para clasificar eventos relacionados con la voz hablada, cantada o la música, muchas bases de datos solo contienen muestras de este tipo, lo cual no es suficiente para entrenar clasificadores orientados a nuevos usos. Por esta razón, es necesario recurrir a bases de datos que contengan una variedad suficiente de muestras acordes con el destino del modelo. En las bases de datos fuertemente etiquetadas, la etiqueta no solo hace referencia al tipo de evento que representa, sino que también indica el inicio y el fin del evento en el tiempo. Sin embargo, estas bases de datos fuertemente anotadas no son comunes, ya que requieren una cantidad abrumadora de trabajo. Para mitigar la falta de muestras, en algunos casos se ha realizado la mezcla de muestras de distintas bases de datos, lo que aumenta la cantidad de información para entrenar el clasificador. No obstante, esta estrategia no refleja situaciones reales, a pesar de que las etiquetas siguen siendo fiables desde su origen. En la literatura existen algunas bases de datos fuertemente etiquetadas, como Dcase 2013 y TUT Sound Event. DCASE 2013 fue elaborada por Emmanouil Benetos y Mathieu Lagrange [25], dado que no había ninguna pública con las características necesarias para el proyecto, decidieron hacer ellos mismos las grabaciones en distintas zonas de Londres. Las grabaciones fueron realizadas a distintas horas del día y con cuidado de que no hubiera interferencias de otro tipo de eventos que no fueran objetivo del estudio, haciendo ellos mismos el etiquetado. Otro ejemplo de base de datos fuertemente etiquetada es la base TUT Sound Events [26], un subset de la TUT Acoustic Scenes 2016 orientada a los SED. Está dividida en dos entornos comunes, uno en el exterior en una zona residencial y otro interior en una casa. Son las zonas de interés para el desarrollo de un SED orientado a la seguridad y la vigilancia, dispone de 7 clases en la zona residencial y 11 en la doméstica. Por otro lado, en otros modelos se utilizan bases de datos de anotación débil, en las que las muestras solo incluyen el tipo de evento que representan, sin incluir la posición temporal. Este tipo de bases de datos es mucho más común aunque la aparición de AudioSet en 2017, pese a ser también de anotación débil, es de uso general por la cantidad de clases que contiene, lo que ha supuesto un antes y un después. Las bases de datos más comunes son:

1. UrbanSound8K[27] lanzada en 2014 y que está orientada a sonidos urbanos provenientes de Freesound, que cuenta con 1302 muestras revisadas sumando 18.5 horas de audio para muestras que varían desde los 1-2s hasta los 30s para sonidos contínuos como pueden ser martillos hidráulicos.

2. ESC-50[28], lanzada en 2015, utiliza sonidos provenientes de Freesound al igual que UrbanSound8k, sin embargo esta está balanceada. Consiste en 2000 grabaciones etiquetadas de 50 clases con al menos 40 clips por clase. Estas clases se agrupan en 5 grupos:
   - **a)** Sonidos de animales
   - **b)** Sonidos naturales y del agua
   - **c)** Sonidos humanos no relacionados con el habla
   - **d)** Sonidos domésticos o del interior
   - **e)** Sonidos urbanos o del exterior

   Una de las posibles debilidades de esta base de datos es la escasez de muestras por clase debido al alto coste de la anotación manual de los archivos de sonido.
3. CHiME-home[29], lanzado en 2015, cuenta con 6.8 horas de audio binaural que se puso a disposición del proyecto Computational Hearing in Multisource Environments (CHiME). Las grabaciones se realizaron en un adosado inglés de la era victoriana. La anotación fue realizada por tres estudiantes de ingeniería que se encargaron de las 6138 muestras con 9 etiquetas distintas:
   - **a)** Habla infantil
   - **b)** Habla de adulto masculino
   - **c)** Habla de adulto femenino
   - **d)** Videojuego/TV
   - **e)** Sonidos de percusión, ej. choque, golpe, pasos
   - **f)** Ruido de banda ancha, ej. electrodomésticos
   - **g)** Otros sonidos identificables
   - **h)** Silencio/ruido de fondo
   - **i)** Fragmento marcado (sonidos no identificables o ambiguos)
4. AudioSet[30], es una base de datos creada a partir de segmentos etiquetados de vídeos de YouTube, estructurados en un archivo CSV que contiene el enlace al vídeo, el tiempo de inicio y el tiempo final de la muestra. Estos "timestamps" no indican cuándo ocurre el evento, sino que nos indican en qué parte del vídeo se toma la muestra para la base de datos. AudioSet cuenta con más de 2 millones de muestras abarcando más de 500 clases de sonidos, y está estructurada en una ontología jerárquica. Sin embargo no es fácil de utilizar debido a los términos y condiciones de YouTube relacionados con el copyright ya que no permite la descarga de las muestras para el uso fuera de línea por ejemplo o la pérdida de ellas por el borrado de los vídeos. La describiremos en detalle en la sección 2.1.
5. FSD50K[31], es una base de datos abierta que contiene 51,000 muestras de audio que suponen un total de 100 horas manualmente etiquetadas basándose en 200 clases de la ontología de AudioSet. Esta es la base de datos que utilizaremos en este trabajo debido a su disponibilidad.

2 Metodología

En este capítulo en primer lugar se detallará la base de datos que hemos utilizado, a continuación explicaremos el proceso de extracción de características, desde el preprocesado de las muestras y las etiquetas, hasta la obtención de los embeddings y finalmente detallaremos el entrenamiento de los clasificadores y el proceso de validación.

2.1 FSD50K

Esta base de datos, a diferencia de Audioset, proporciona los archivos sonoros en formato .wav, característica determinante para usarla en este trabajo, además de seguir la ontología de Audioset, mostrada en la figura 2.1. La base de datos está dividida en un conjunto para el desarrollo y entrenamiento de los sistemas y otro para la evaluación del sistema desarrollado.

Figura 2.1 Ontología Audioset[30].

Para la obtención de los archivos hay que dirigirse a la página web de la base de datos[1], en la que podremos encontrar las carpetas que lo componen tal y como se muestra en la figura 2.2 con la división de evaluación y desarrollo propuesta.
La base de datos contiene dos conjuntos de archivos de audio, denominados dev, que contiene 40967 muestras, y eval, que contiene 10232 muestras, así como las respectivas etiquetas o ground truth para cada conjunto. La duración de las muestras varía entre 0.3s y 30s. El conjunto de desarrollo (dev) es el que utilizaremos para entrenar el clasificador. En la ontología AudioSet una misma muestra puede tener más de una etiqueta, pudiendo ser además de distinto nivel. En el caso de las muestras que corresponden a la sirena de un coche de policía tiene dos etiquetas con distinta jerarquía, la primera es de 5 niveles y corresponde a *Sound of Things* > *Vehicle* > *Motor vehicle (road)* > *Emergency vehicle* > *Police car (Siren)*, mientras que la segunda etiqueta que sería la correspondiente a la alarma tiene 4 niveles que son *Sounds of Things* > *Alarm* > *Siren* > *Police car (Siren)*.

[1] https://zenodo.org/records/4060432

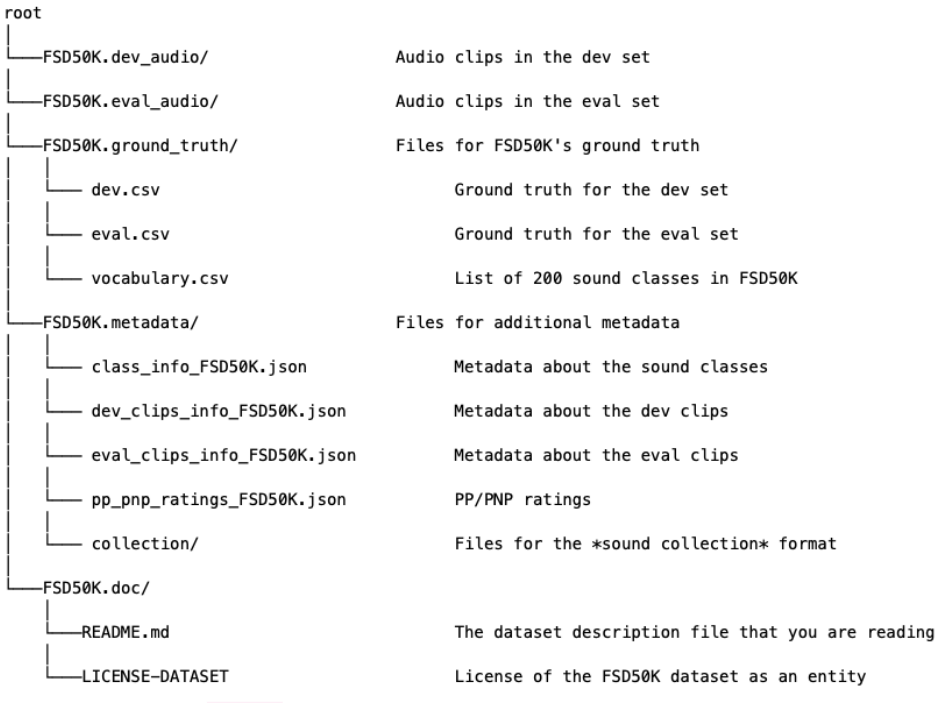

Figura 2.2 Estructura de directorios del Dataset tal y como describen en su web.

La estructura del archivo .csv del ground truth por filas, tal y como se muestra en la figura 2.3, es la siguiente:

- *fname*: Es el nombre del archivo sin extensión, si el archivo es 60354.wav en esa columna aparecerá *60354*.
- *labels*: Contiene las etiquetas del archivo.
- *mids*: Contiene los identificadores de Freesound siguiendo la ontología de Audioset.
- *split*: Indica a que split pertenece la muestra, ya sea a dev o eval.

fname	labels	mids	split
64760	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
16399	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
16401	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
16402	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
16404	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
345111	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	val
64761	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
268259	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
64762	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
160826	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	val
40515	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
40516	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
40517	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
420945	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	val
420946	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	val
420947	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	val
86100	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	val
64741	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
40523	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
64743	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
64744	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
40525	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
51319	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	val
64746	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
5318	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
4258	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
4259	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
4260	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
4261	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
4262	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
4263	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
4264	Electric_guitar,Guitar,Plucked_string_instrument,Musical_instrument,Music	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train

Figura 2.3 Fragmento del ground truth de FSD50K.

Aunque la parte de evaluación de las muestras tiene etiquetas verificadas, existen ambigüedades que dificultan la clasificación precisa de algunas de ellas. Como hemos expuesto previamente, en la ontología de AudioSet ya aparecen en algunas muestras con distintos niveles y etiquetas de ámbito completamente opuesto. Además, al inspeccionar el archivo .csv con las etiquetas, hemos descubierto que el uso de ellas no era jerárquico. Estos problemas se han plasmado en la trasposición realizada de la ontología en FSD50K, pese a que los identificadores están situados correctamente, dando lugar a que algunas muestras sean consideradas ambiguas al escucharlas. Al principio, se intentó utilizar muestras de sonido con una sola etiqueta, es decir, tomar en cuenta todas las muestras que tuvieran una sola etiqueta, ya fuera alarma, teléfono, campana, etc. A cada una se le asignó una etiqueta numérica, no obstante, este método resultó ser extremadamente complejo y poco práctico, además el tiempo necesario para procesar las muestras era demasiado largo. En su lugar, se decidió entrenar un sistema SED para la detección de eventos tipo "Alarma" que pertenece al grupo "Sonido de las cosas" de la ontología de AudioSet. En este caso incluiremos las muestras con etiqueta "Alarm" y los subtipos dentro de la ontología de AudioSet que detallaremos a continuación. No hemos podido ser más precisos debido a las inconsistencias del etiquetado.

Según la ontología de AudioSet existen las siguientes subcategorías dentro de las Alarmas:

- Telephone
- Alarm Clock
- Siren
- Doorbell
- Buzzer
- Smoke Detector, Smoke Alarm
- Fire Alarm
- Vehicle Horn, car horn, honking
- Bicycle bell
- Air horn, truck horn
- Foghorn
- Whistle

Aunque FSD50K emplea la ontología Audioset, presenta algunas diferencias e inconsistencias en el etiquetado. Por ejemplo, la muestra 176066 tiene asignadas las etiquetas "Telephone, Typing, Alarm, Domestic sounds and home sounds", pero al escucharla, solo se percibe el sonido de un teclado virtual de smartphone. Tras intentar sin éxito separar el conjunto de muestras de Alarmas en niveles jerárquicos inferiores de la ontología para aumentar el conjunto de datos, se tomó la decisión de incorporar todas las muestras que incluyeran la etiqueta "Alarm", independientemente de los niveles adicionales que pudieran tener. Aunque puede introducir más inconsistencias en los datos, aumenta el tamaño del conjunto de entrenamiento. Esta decisión se basa en la hipótesis de que un conjunto de datos más grande, aun con algunas inconsistencias, proporcionará mejores resultados que uno pequeño. La diversidad adicional podría incluso mejorar la capacidad del modelo para manejar variaciones y ambigüedades en entornos reales. El algoritmo que hemos implementado para diseñar el sistema SED está expuesto en la figura 2.4 en forma de diagrama de flujo. El proceso tiene ligeras variaciones, en los dos grupos tenemos que asignar la etiqueta a cada muestra de audio siguiendo las equivalencias del ground_truth de forma manual, ya que MATLAB nos da la opción únicamente del agrupamiento de clases por carpetas, después contaremos los frames o segmentos para conocer el número de ellos que corresponden al evento "Alarm" y poder balancear los datos de entrenamiento. En el caso del grupo de evaluación el proceso es exactamente igual, los segmentos o frames se cuentan para balancear los datos aunque estos no se utilizan para entrenar.

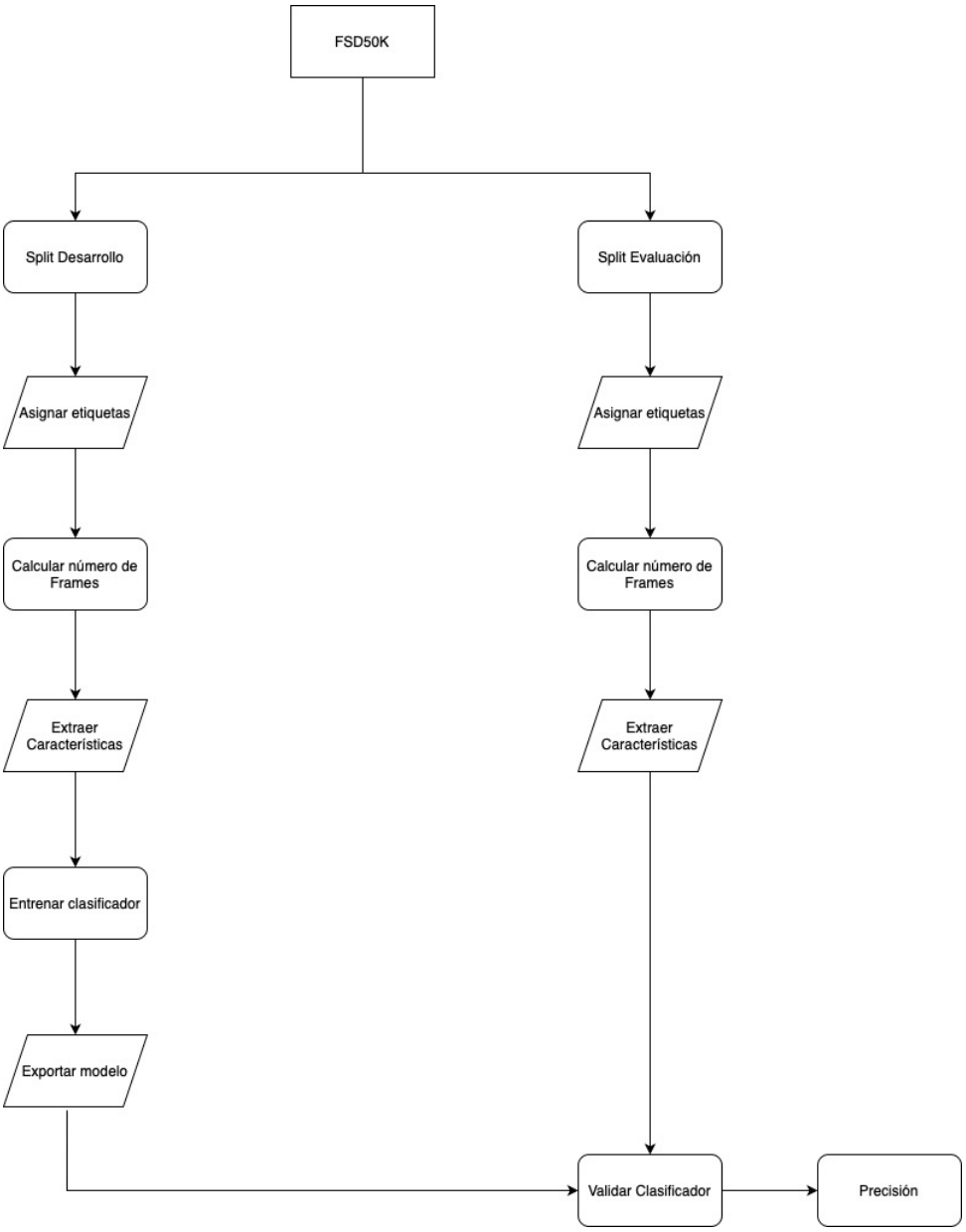

Figura 2.4 Diagrama de flujo del algoritmo implementado..

2.2 Preprocesado de las muestras

MATLAB proporciona el objeto "audioDataStore" dentro del AudioToolbox, una herramienta diseñada para almacenar muestras de sonido junto con sus etiquetas. Una de sus funciones más útiles es la capacidad de asignar etiquetas a los archivos de audio basándose en la estructura de subcarpetas. Sin embargo, en nuestro caso, esta funcionalidad no es aplicable, ya que las etiquetas no están organizadas jerárquicamente en el sistema de archivos, sino que se encuentran en un archivo .csv del ground truth. Para adaptar el "audioDataStore" a nuestra estructura de datos, fue necesario desarrollar un código personalizado en MATLAB. Este script lee cada archivo de audio individualmente y busca sus etiquetas correspondientes en el archivo .csv. Una vez encontradas, asigna estas etiquetas al archivo dentro del datastore. Este proceso, aunque más complejo que usar la función predeterminada de MATLAB, nos permite aprovechar el "audioDataStore" mientras mantenemos la integridad de la estructura de los ficheros. Una ventaja adicional de este método es la capacidad de guardar el datastore generado para usos futuros. Esto es particularmente útil porque la generación del datastore es un proceso bastante lento. El tiempo requerido para leer cada archivo, buscar sus etiquetas y asignarlas puede variar significativamente dependiendo de las características del hardware. Al guardar el datastore una vez generado, evitamos tener que repetir este proceso cada vez que trabajamos con los datos, lo que resulta en un ahorro de tiempo notable, especialmente en equipos con recursos más limitados.

Código 2.1 Código que asigna las etiquetas.

```
% Specify the path to the directory containing the audio files
tStart=tic;
dataDir = 'C:\FSD50K\FSD50K.dev_audio';

% Create an audio datastore for the audio files
ads = audioDatastore(dataDir, 'IncludeSubfolders', true,'FileExtensions'
    ,'.wav');

% Specify the path to the CSV file containing the labels
labelsFile = 'C:/FSD50K/FSD50K.ground_truth/dev.csv';

% Load the labels from the CSV file
labelsTable = readtable(labelsFile, 'Delimiter', ',');
labelsTable(:,3)=[];
% Convert the table to a cell array
labelsCell = table2cell(labelsTable);

% Match the labels with the filenames in the audio datastore
numFiles = length(ads.Files);
nombres = ads.Files;
C = cell(1,numFiles);
C(:)={'Unknown'};
ads.Labels=C;
for i = 1:numFiles
    archivo = split(nombres(i),dataDir);
    nombres(i)=archivo(2,1);
    archivo = split(nombres(i),'.');
    nombres(i)=archivo(1,1);
```

```
end
for i = 1:numFiles
    labelIdx=0;
    j = 1;
    while (labelIdx == 0 && j<=numFiles)
        if(strcmp(string(labelsCell{j,1}),string(nombres(i))))
            labelIdx=j;
        end
        j=j+1;
    end

    % If label is found, associate it with the audio file
    if labelIdx~=0
       label = labelsCell{labelIdx, 2};
       ads.Labels{i} = label;
    else
        %If label is not found, you can assign a default label or handle
            it as you wish
        ads.Labels{i} = 'Unknown';
    end
end
```

En el código 2.1 se muestra el código MAtlab que realiza las operaciones descritas anteriormente, donde:

- *dataDir*: Es el directorio donde se encuentran los archivos de las muestras.
- *ads*: Es el audioDataStore donde almacenaremos las muestras con sus etiquetas.
- *labelsFile*: Es el archivo que contiene las etiquetas correspondientes a cada muestra.

En el código mostrado anteriormente se realiza la asignación de etiquetas para el split de desarrollo de FSD50K, para hacerlo con el de evaluación simplemente habría que cambiar *dataDir* y *labelsFile* a los valores correspondientes. La ejecución del código para generar el audioDataStore *ads*, un proceso que implica recorrer individualmente las 40,966 muestras, que como hemos mencionado previamente, es bastante lento. En un portátil equipado con un procesador Intel i5-8300H y 16GB de RAM, este proceso consumió un tiempo cercano a una media hora. Sorprendentemente, al realizarlo en un Mac mini M1 con solo 8GB de RAM, se observó una mejora de rendimiento muy notable, ya que tarda de media un tercio menos que el equipo convencional con arquitectura x86.

2.3 Extracción de embeddings

MATLAB nos ofrece dos modelos preentrenados para extraer los embeddings de las muestras:

- VGGish [32] Extrae características usando un modelo preentrenado con AudioSet para la clasificación de sonidos. Opera con un tamaño mínimo de las muestras de aproximadamente 0.975s sin solapamiento y el número de características es de 128 por vector. En [16] analizan el uso de redes CNN, que se usan para clasificación de imágenes, en el ámbito de la clasificación de eventos sonoros. Las redes que se analizan, entre ellas VGG, demuestran que las características procedentes de estas redes son mucho más eficaces para la clasificación de eventos sonoros en comparación con las características convencionales.
- OpenL3[32] Extrae características mediante un modelo preentrenado en la red L^3 en un subconjunto de AudioSet. Tiene una resolución de 0.1s sin solapamiento y el número de características es de 512 por vector. En [19] evalúan el impacto las decisiones que se pueden tomar a la hora de diseñar y entrenar un SED de la red L^3, siendo el resultado más relevante el mejor rendimiento experimentado frente a VGGish.

Tras comparar el tiempo de procesamiento de ambos modelos, se optó por VGGish para este proyecto. Esta elección se basa en su velocidad significativamente mayor y su rendimiento generalmente superior a OpenL3[32]. Sin embargo, esta decisión presentó un problema importante: muchas muestras en FSD50K tienen una duración inferior a la mínima de 0.975s requerida por VGGish. Para solucionar este problema, optamos por alargar las muestras. Antes de la extracción, se verifica la longitud de cada muestra. Si es inferior a 0.975s, la muestra se duplica las veces necesarias hasta alcanzar la duración mínima requerida, asegurando así la compatibilidad con VGGish sin perder datos ni introducir errores.
Para entrenar un clasificador efectivo, es crucial mantener los datos balanceados. En este caso, se busca equiparar el número de muestras etiquetadas como "Alarm" con aquellas que contienen otras etiquetas. El balanceo de clases previene el sesgo en el modelo, que de otro modo podría inclinarse hacia la clase predominante, ignorando o clasificando erróneamente las menos representadas. Para lograr este equilibrio, el primer paso es contar cuántas muestras contienen la etiqueta "Alarm". Desarrollamos un script en MATLAB para esta tarea, su función es recorrer el conjunto de datos, identificando y contando las muestras que incluyen "Alarm" entre sus etiquetas. Este proceso, aunque parece simple, es fundamental para establecer la línea base con la que se compararán otras clases. Una vez obtenido este recuento, seleccionamos un número similar de muestras que no contengan la etiqueta "Alarm". Dado que la base de datos es de etiquetado débil, no sabemos en qué momento ocurre el evento sonoro, por lo que hemos decidido aplicar esta etiqueta a todos los frames extraídos de esa muestra. Esta selección será aleatoria dentro de la base de muestras de FSD50K, asegurando una representación equitativa de diversas categorías. El resultado es un conjunto de entrenamiento equilibrado, donde la presencia o ausencia de alarmas tiene igual peso. Este método no solo mejora la precisión del modelo al evitar sesgos, sino que también aumenta su generalización. En el código 2.2 se muestra cómo se ha utilizado una función *calcula* para aumentar la duración de las muestras antes de realizar la extracción de los embeddings. Además cuenta el número de frames que tienen todas las muestras con la etiqueta Alarma que será necesario en el siguiente paso para hacer balanceo del entrenamiento.

Código 2.2 Código que cuenta el número de frames en Alarmas.

```
jukdownloadFolder = fullfile(tempdir,"VGGishDownload");
loc = websave(downloadFolder,"https://ssd.mathworks.com/supportfiles/
    audio/vggish.zip");
VGGishLocation = tempdir;
unzip(loc,VGGishLocation)
addpath(fullfile(VGGishLocation,"vggish"))
load("ADSEval.mat");
reset(ads);
numFiles=length(ads.Files);
contador=0;
for i = 1:numFiles
    Alarma=false;
    [audioIn,fs] = read(ads);
    labels=split(string(fs.Label),",");
    if length(labels)>=1
        for n=1:length(labels)
            if(strcmp(string("Alarm"),string(labels(n))))
                contador=contador+1;
                embeddings=calcula(audioIn,fs.SampleRate);
                for j=1:size(embeddings,1)
                    if j>1 && j<size(embeddings,1)
                        contador=contador+1;
                    end
                end
            end
        end
    end
end

function embed=calcula(audioIn,SampleRate)
    N=length(audioIn);
    slength=N/SampleRate;
    if slength<=0.975
        while slength<=0.975
            audioIn=cat(1,audioIn,audioIn);
            N=length(audioIn);
            slength=N/SampleRate;
        end
    end
    embed = vggishEmbeddings(audioIn,SampleRate);
end
```

Donde:

- *numFiles*: Es el número de muestras contenidas en el audioDataStore.
- *contador*: Es la variable que cuenta el número de frames.
- *calcula*: Es la función que aumenta la duración de las muestras.

Sin embargo, la propuesta inicial de balanceo basada en muestras completas resultó ser poco útil. El problema surge al considerar la división de las muestras de audio: los frames. En el ámbito del procesamiento de señales de audio, cada muestra se compone de múltiples frames, segmentos temporales breves que capturan las características acústicas en instantes específicos. Una muestra larga de una alarma de incendio, por ejemplo, contendrá más frames que una breve alerta de notificación en un teléfono. Esta disparidad en la cantidad de frames distorsiona el equilibrio que se buscaba inicialmente a nivel de muestras completas. Para lograr un balanceo verdaderamente preciso, es necesario adaptarlo a nivel de frames. Esto implica descomponer cada muestra en sus frames constituyentes y luego igualar el conjunto de datos basándose en estas unidades. Por ejemplo, si las muestras de "Alarm" suman 10,000 frames, se seleccionarán otras muestras cuya suma de frames también se aproxime a 10,000. Esta adaptación, aunque más compleja, garantiza un balanceo mucho más preciso.

En el código 2.3 generaremos la tabla de características ya preparada para realizar el entrenamiento en el Classification Learner.

Código 2.3 Código que extrae y formatea las características de las muestras.

```
downloadFolder = fullfile(tempdir,"VGGishDownload");
loc = websave(downloadFolder,"https://ssd.mathworks.com/supportfiles/
    audio/vggish.zip");
VGGishLocation = tempdir;
unzip(loc,VGGishLocation)
addpath(fullfile(VGGishLocation,"vggish"))
load("ADSEval.mat");
reset(ads);
numFiles=length(ads.Files);
contador=0;
NoAlarma=0;
Alarmas=11479; %19635 para dev
Validacion={};
etiquetasVal=[];
contadorVal=0;
eV={};
etiquetas=[];
for i = 1:numFiles
    Alarma=false;
    [audioIn,fs] = read(ads);
    labels=split(string(fs.Label),",");
    if length(labels)>1
        for n=1:length(labels)
            if(strcmp("Alarm",string(labels(n))))
                Alarma=true;
                contador=contador+1;
                embeddings=calcula(audioIn,fs.SampleRate);
                contadorVal=contadorVal+1;
                Validacion{contadorVal}=embeddings;
                etiquetasVal(contadorVal)=1;
                for j=1:size(embeddings,1)
                    etiquetas(contador)=1;
                    eV{contador}=embeddings(j,:);
```

```
                if j>1 && j<size(embeddings,1)
                    contador=contador+1;
                end
            end
        end
     end
end
    if Alarma==false && NoAlarma<=Alarmas
        contador=contador+1;
        embeddings=calcula(audioIn,fs.SampleRate);
        NoAlarma=NoAlarma+1;
        contadorVal=contadorVal+1;
        Validacion{contadorVal}=embeddings;
        etiquetasVal(contadorVal)=0;
        for j=1:size(embeddings,1)
            etiquetas(contador)=0;
            eV{contador}=embeddings(j,:);
            if j>1 && j<size(embeddings,1)
               contador=contador+1;
               NoAlarma=NoAlarma+1;
            end
        end
    else
    if(strcmp("Alarm",string(labels)))
        contador=contador+1;
        embeddings=calcula(audioIn,fs.SampleRate);
        contadorVal=contadorVal+1;
        Validacion{contadorVal}=embeddings;
        etiquetasVal(contadorVal)=1;
        for j=1:size(embeddings,1)
            etiquetas(contador)=1;
            eV{contador}=embeddings(j,:);
            if j>1 && j<size(embeddings,1)
               contador=contador+1;
            end
        end
    else
        if NoAlarma<=Alarmas
            contador=contador+1;
            embeddings=calcula(audioIn,fs.SampleRate);
            NoAlarma=NoAlarma+1;
            contadorVal=contadorVal+1;
            Validacion{contadorVal}=embeddings;
            etiquetasVal(contadorVal)=0;
            for j=1:size(embeddings,1)
                etiquetas(contador)=0;
                eV{contador}=embeddings(j,:);
                if j>1 && j<size(embeddings,1)
                   contador=contador+1;
                   NoAlarma=NoAlarma+1;
```

```
                end
            end
        end
     end
    end
end
mat=cell2mat(eV.');
TablaMulti=table(mat,etiquetas.');

function embed=calcula(audioIn,SampleRate)
    N=length(audioIn);
    slength=N/SampleRate;
    if slength<=0.975
        while slength<=0.975
            audioIn=cat(1,audioIn,audioIn);
            N=length(audioIn);
            slength=N/SampleRate;
        end
    end
    embed = vggishEmbeddings(audioIn,SampleRate);
end
```

El código 2.3 es una adaptación del contador que hemos visto previamente en el código 2.2, realizando el cálculo de los embeddings de muestras aleatorias que no son alarmas, además se observa como separa los frames y les asigna a cada uno la etiqueta correspondiente. En este código se realiza para el split de desarrollo aunque en el de evaluación es muy parecido a excepción del número de frames *Alarmas* y el datastore utilizado. El numero de frames total es de 38931. En el código 2.3 se muestra el proceso completo donde:

- *Alarmas*: Es el número de frames total de todas las muestras que contienen la etiqueta.
- *NoAlarma*: Es el contador de frames aleatorios que no corresponden a ninguna Alarma.
- *Validacion*: Es un array en formato cell que contiene las muestras completas con todos sus frames.
- *etiquetasVal*: Es el array que contiene las etiquetas correspondientes a cada muestra completa, para realizar la validación posteriormente.

2.4 Extracción de características MFCC

Para comparar los resultados de nuestro trabajo, vamos a diseñar un SED alternativo utilizando características MFCC en la que utilizaremos una versión modificada del código utilizado para la extracción de los embeddings tal y como se muestra en el código 2.4.

Código 2.4 Código que extrae las características físicas.

```
load("ADSEvalWin.mat");
reset(ads);
numFiles=length(ads.Files);
contador=0;
NoAlarma=1;
Alarmas=0; %6097; eval MFCC %10617; dev MFCC %11479 para eval Vgg;
    %19635 para dev Vgg
Validacion={};
etiquetasVal=[];
contadorVal=0;
eV={};
etiquetas=[];
for i = 1:numFiles
    Alarma=false;
    [audioIn,fs] = read(ads);
    labels=split(string(fs.Label),",");
    if length(labels)>1
        for n=1:length(labels)
            if(strcmp("Alarm",string(labels(n))))
                Alarma=true;
                contador=contador+1;
                %Alarmas=Alarmas+1;
                embeddings=calcula(audioIn,fs.SampleRate);
                contadorVal=contadorVal+1;
                Validacion{contadorVal}=embeddings;
                etiquetasVal(contadorVal)=1;
                for j=1:size(embeddings,1)
                    etiquetas(contador)=1;
                    eV{contador}=embeddings(j,:);
                    if j>1 && j<size(embeddings,1)
                        contador=contador+1;
                        %Alarmas=Alarmas+1;
                    end
                end
            end
         end
    end
        if Alarma==false && NoAlarma<=Alarmas
            contador=contador+1;
            embeddings=calcula(audioIn,fs.SampleRate);
            NoAlarma=NoAlarma+1;
            contadorVal=contadorVal+1;
```

```
            Validacion{contadorVal}=embeddings;
            etiquetasVal(contadorVal)=0;
            for j=1:size(embeddings,1)
                etiquetas(contador)=0;
                eV{contador}=embeddings(j,:);
                if j>1 && j<size(embeddings,1)
                   contador=contador+1;
                   NoAlarma=NoAlarma+1;
                end
            end
        else
        if(strcmp("Alarm",string(labels)))
            contador=contador+1;
            embeddings=calcula(audioIn,fs.SampleRate);
            contadorVal=contadorVal+1;
            Validacion{contadorVal}=embeddings;
            etiquetasVal(contadorVal)=1;
            for j=1:size(embeddings,1)
                etiquetas(contador)=1;
                eV{contador}=embeddings(j,:);
                if j>1 && j<size(embeddings,1)
                   contador=contador+1;
                end
            end
        else
            if NoAlarma<=Alarmas
                contador=contador+1;
                embeddings=calcula(audioIn,fs.SampleRate);
                NoAlarma=NoAlarma+1;
                contadorVal=contadorVal+1;
                Validacion{contadorVal}=embeddings;
                etiquetasVal(contadorVal)=0;
                for j=1:size(embeddings,1)
                    etiquetas(contador)=0;
                    eV{contador}=embeddings(j,:);
                    if j>1 && j<size(embeddings,1)
                       contador=contador+1;
                       NoAlarma=NoAlarma+1;
                    end
                end
            end
         end
        end
end

%{
function embed=calcula(audioIn,SampleRate)
coeffs = mfcc(audioIn, SampleRate); % Extraer MFCC
deltas = diff(coeffs);   % Calcular coeficientes delta ()
delta_deltas = diff(deltas); % Calcular delta-delta ()
```

```
% Calcular promedios
mfcc_mean = mean(coeffs, 1);      % Media de los MFCC
delta_mean = mean(deltas, 1);     % Media de los Delta
delta_delta_mean = mean(delta_deltas, 1); % Media de los Delta-Delta
% Combinar características en un solo vector
embed = [mfcc_mean, delta_mean, delta_delta_mean];
end
%}
mat=cell2mat(eV.');
TablaMulti=table(mat,etiquetas.');
balanceo = sum(etiquetas)/width(etiquetas)
tabla=table(Validacion.',etiquetasVal.');

function embed=calcula(audioIn,SampleRate)
    N=length(audioIn);
    slength=N/SampleRate;
    if slength<=0.975
        while slength<=0.975
            audioIn=cat(1,audioIn,audioIn);
            N=length(audioIn);
            slength=N/SampleRate;
        end
    end
    matrix = mfcc(audioIn,SampleRate);
    % Asume que 'matrix' es tu matriz de características MFCC (
        dimensiones: frames x coeficientes)
    [num_frames, num_coefs] = size(matrix); % num_frames = número de
        vectores de características, num_coefs = número de coeficientes
        MFCC

    % Tamaño del grupo de vectores
    group_size = 97;

    % Calcula el número de grupos completos
    num_groups = floor(num_frames / group_size);

    % Inicializa la matriz para almacenar las medias de cada grupo
    matrix_mean = zeros(num_groups, num_coefs); % Almacenará las medias
        de cada grupo de 97 vectores

    % Calcula la media de cada grupo de 97 vectores
    for i = 1:num_groups
        start_idx = (i-1)*group_size + 1;
        end_idx = start_idx + group_size - 1;

        % Extrae el grupo y calcula la media
        group = matrix(start_idx:end_idx, :);
        matrix_mean(i, :) = mean(group, 1); % Calcula la media de las
            filas para cada columna
    end
```

```
    % Si quedan vectores adicionales, calcula la media del grupo
        incompleto
    if mod(num_frames, group_size) ~= 0
        remaining_group = matrix(num_groups * group_size + 1:end, :);
        matrix_mean(num_groups + 1, :) = mean(remaining_group, 1);
    end

    % 'matrix_mean' contiene las medias de los grupos de 97 vectores
    embed=matrix_mean;
end
```

Para que los SED sean comparables y se pueda computar en un tiempo razonable, hemos reducido el número de vectores de características MFCC para que cada uno correspondieran a una ventana de tiempo similar haciendo la media de cada grupo de 97 vectores. Esta modificación se ha realizado directamente en la función que calcula los vectores por lo que este código es prácticamente idéntico al anterior como hemos mencionado previamente.

2.5 Classification Learner

Para crear un modelo de clasificación utilizaremos una herramienta de MATLAB llamada Classification Learner, que permite seleccionar características, especificar métodos de validación, entrenar modelos y evaluar resultados. Este acepta vectores fila de datos en los que puede haber una etiqueta por vector, puede ser binario o multiclase. En nuestro caso se trata de un sistema binario ya que queremos que detecte si en la muestra que vamos a analizar aparece un evento sonoro de tipo alarma. Utilizaremos 38931 vectores de dimensión 128, que es el número de características, y la etiqueta si contienen alarma es "1" y si no la tienen es "0". Hemos utilizado el split dev para realizar el entrenamiento, utilizando los métodos estadísticos agrupados en *All Quick-To-Train* mediante los algoritmos:

- Árboles de decisión (Tree)
- K-Nearest Neighbour
- Efficient Logistic Regression
- Efficient Linear SVM

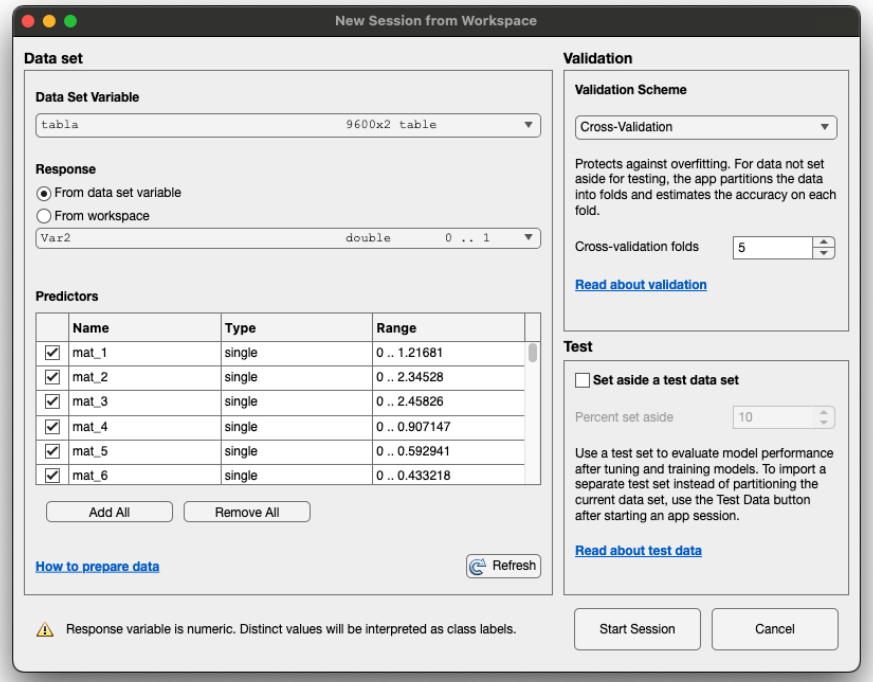

Figura 2.5 Introducción de datos en el Classification Learner.

Una vez introducidos los datos la aplicación permite elegir el método de validación k-fold, la validación k-fold es una técnica utilizada en machine learning para evaluar el rendimiento de un modelo. Consiste en dividir los datos en k subconjuntos o "folds". El modelo se entrena k veces, cada vez usando k-1 folds para entrenar y el fold restante para validar. y permite separar una parte de los datos para hacer tests. Después de seleccionar las características deseadas procedemos a iniciar la sesión.

Una vez hemos la sesión podremos elegir qué metodos utilizar para entrenar los modelos, estamos ante una aplicación muy potente que nos facilitará la comparación de resultados y el testeo. En este caso utilizaremos los entrenamientos rápidos ya que ofrecen una precisión alta con un coste computacional relativamente bajo, este proceso completo se encuentra en el diagrama de flujo de la figura 2.6.

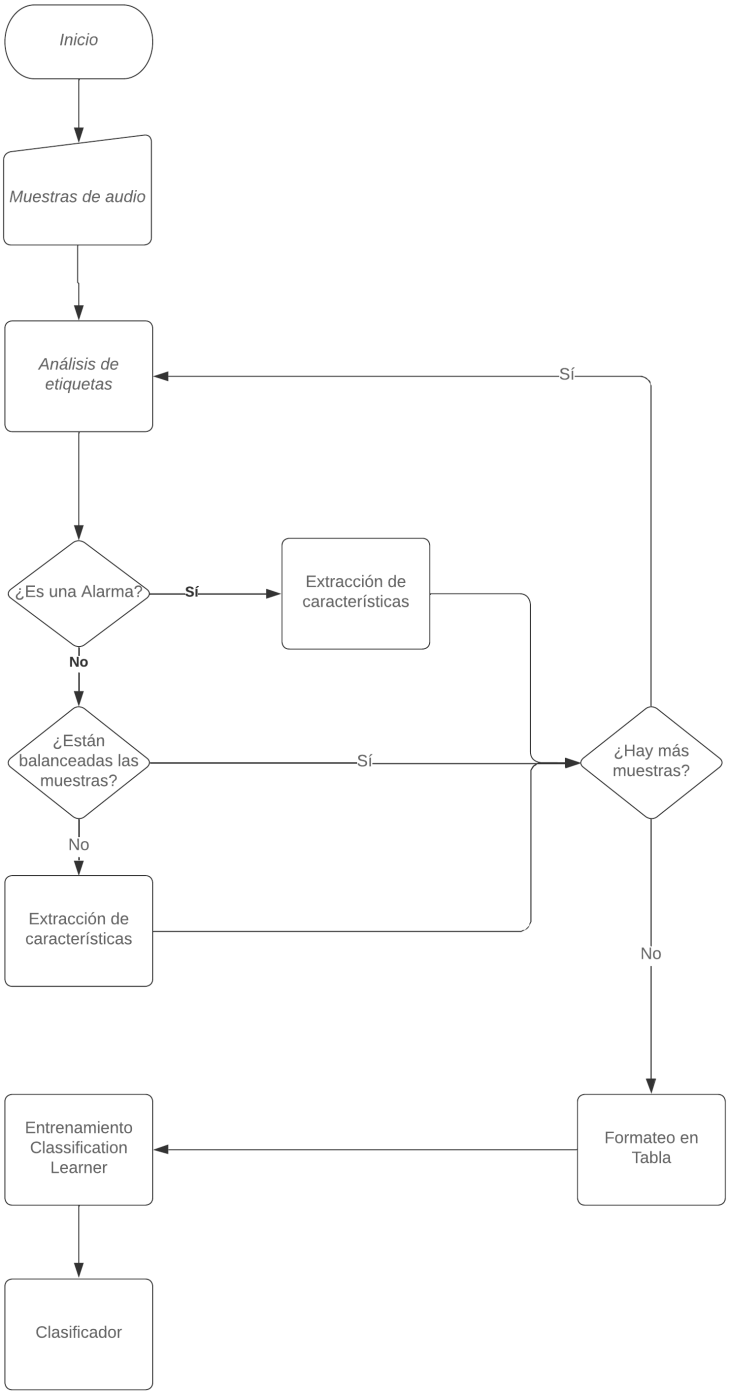

Figura 2.6 Diagrama de flujo del proceso de entrenamiento.

En la figura 2.7 se puede distinguir la matriz de confusión al realizar un entrenamiento de todos los métodos. En este caso el más preciso fue el *KNN subspace*, un conjunto de clasificadores y podemos ver la matriz de confusión de la validación. La matriz de confusión es una tabla que se utiliza para evaluar el rendimiento de un modelo de clasificación. Muestra las cantidades de predicciones correctas e incorrectas. Esto permite analizar en detalle la precisión y los errores del modelo. Después de realizar el entrenamiento podemos proceder al proceso de testeo en el que utilizaremos el split eval del dataset para verificar la precisión del modelo entrenado con las muestras designadas para la evaluación.

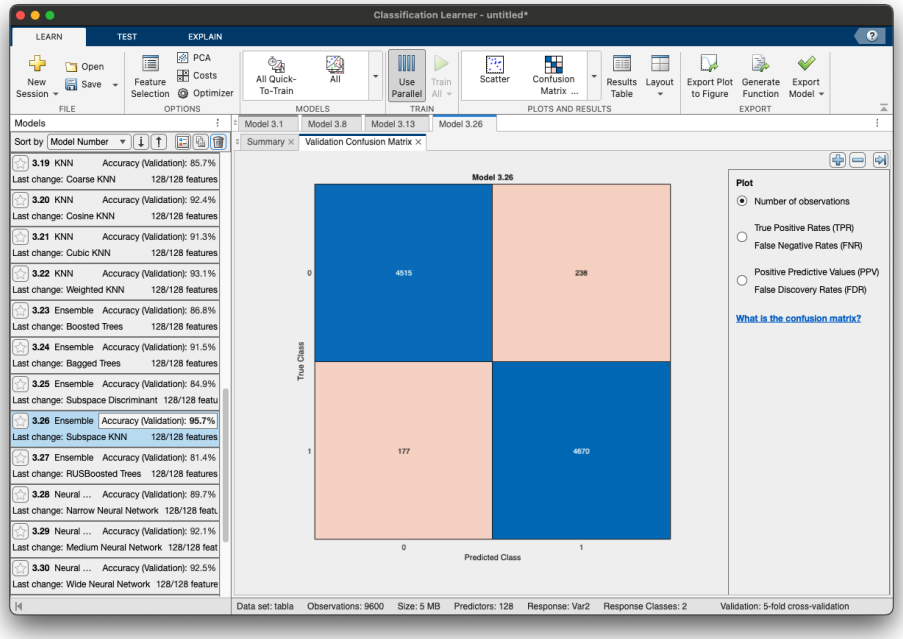

Figura 2.7 Matriz de confusión en un entrenamiento del Classification Learner.

2.6 Validación del modelo

Debido a que hemos separado las muestras en frames debemos volver a validar estos en conjunto, es decir, evaluar todos los frames que corresponden a una misma muestra, ya que de forma individual puede dar positivos tan solo en una parte de la muestra. En este apartado presentaremos el código que hemos realizado y las decisiones que hemos tomado para realizarlo debido a la falta de anotación fuerte.

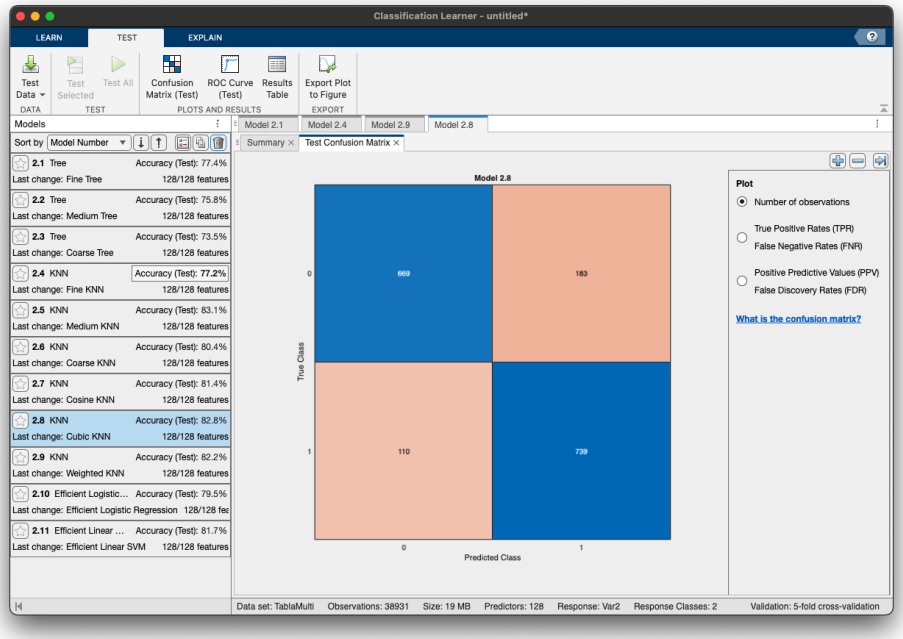

Figura 2.8 Herramienta de testeo del Classification Learner.

En la figura 2.8 se aprecia la herramienta de testeo del classification learner, esta nos permite utilizar otro set de datos distinto al de entrenamiento para comprobar la precisión del clasificador. En este caso utilizaremos el conjunto *eval* completo de FSD50K, que contiene 10232 muestras. Para realizar esa evaluación del modelo entrenado, tenemos el código 2.4 al que podremos realizar algún ajuste del umbral para calcular la precisión. El diagrama de flujo en la figura 2.9 describe el proceso.

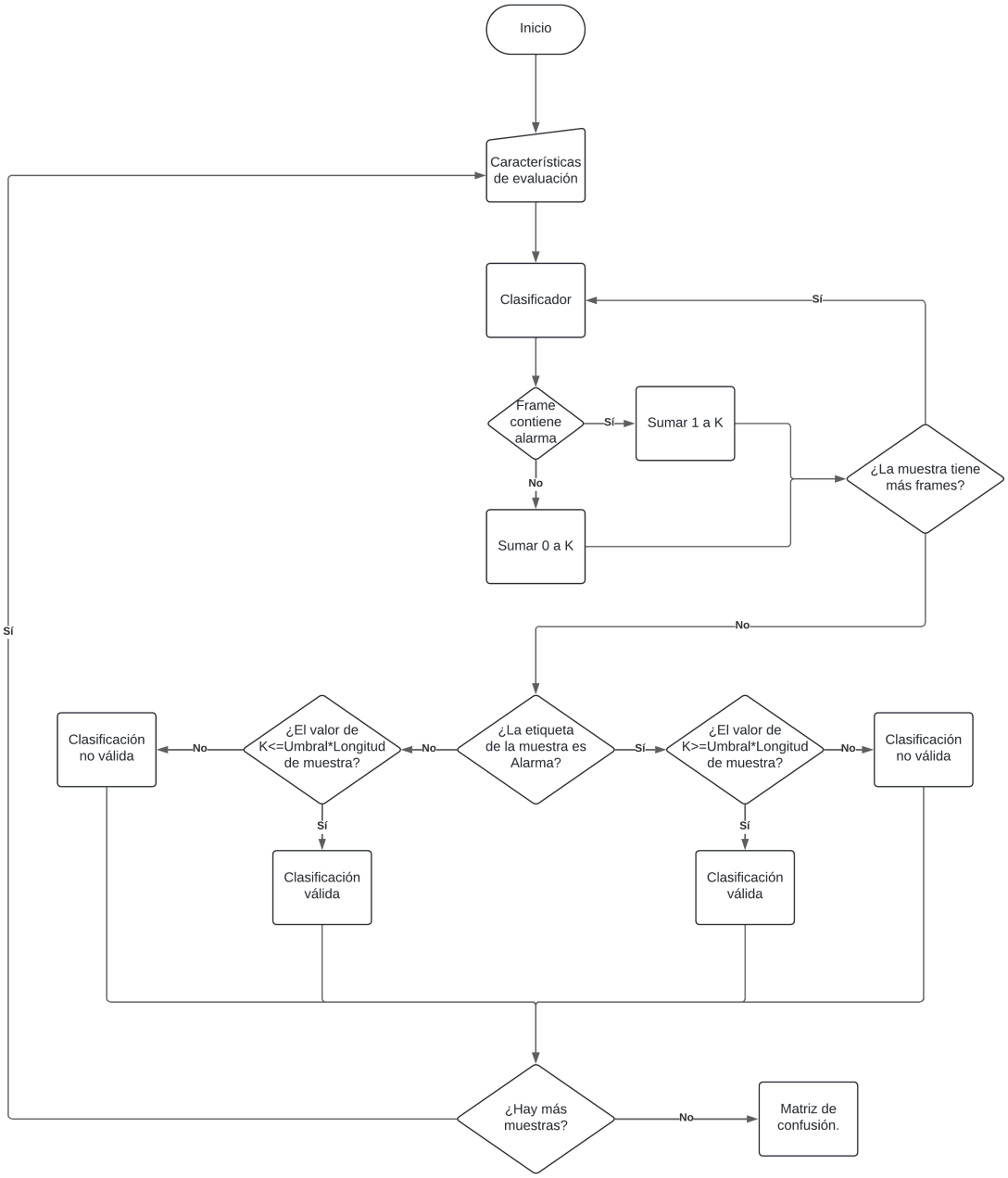

Figura 2.9 Diagrama de flujo del proceso de evaluación.

Código 2.5 Código que valida el resultado del clasificador con muestras completas.

```
load("TablaEval.mat");
load("KNN.mat");
sensibilidad=0.5;
k=0;
valido=[height(tabla)];
predictedLabels=zeros(height(tabla),1);
trueLabels=tabla{:,2};
for i=1:height(tabla)
    k=0;
    embedding=cell2mat(tabla{i,1});
    etiqueta=tabla{i,2};
    for j=1:height(embedding)
        mat=embedding(j,:);
        tt=table(mat,etiqueta);
        k=k+KNN.predictFcn(tt);
    end
    if etiqueta == 1
        if k>=sensibilidad*height(embedding)
            valido(i)=true;
            predictedLabels(i)=1;
        else
            valido(i)=false;
            predictedLabels(i)=0;
        end
    else
        if k<=sensibilidad*height(embedding)
            valido(i)=true;
             predictedLabels(i)=0;
        else
            valido(i)=false;
             predictedLabels(i)=1;
        end
    end
end
figure;
cm = confusionchart(trueLabels,predictedLabels);
beep;
```

Lo primero que realizamos es cargar la tabla con las muestras con todos sus frames, además de la función que utilizaremos para realizar la clasificación. Esta función la obtendremos del classification learner al exportar el modelo entrenado con el split de desarrollo de la base de datos. Después separaremos los frames de cada muestra, realizaremos la clasificación frame a frame y sumaremos el resultado individual de cada uno, que al ser un valor booleano se puede comparar con la longitud de la muestra y así usar el umbral para verificar la validez. Este umbral de decisión indica el porcentaje de frames que tienen que coincidir en una muestra con la etiqueta asignada por el clasificador para que se considere una clasificación válida hemos seleccionado un umbral del 50 % para evitar falsos positivos. Al final calculamos la matriz de confusión mediante la función de MATLAB *confusionchart* y la mostramos.

3 Resultados y conclusiones

En este apartado vamos a exponer los resultados del estudio realizado, detallando la precisión obtenida según el método estadístico elegido. Realizaremos la comparación entre la validación por frames y por muestras completas y explicaremos los problemas encontrados y las soluciones aplicadas. Expondremos las precisiones de los distintos modelos estadísticos del clasificador y justificaremos la decisión tomada con respecto al modelo elegido.

3.1 Resultados del entrenamiento del clasificador

En este apartado expondremos los resultados obtenidos durante el desarrollo del SED, en la fase en la que seleccionaremos los clasificadores basados en su rendimiento clasificando los frames de las muestras. Para realizar estas tareas hemos utilizado la herramienta Classification Learner de Matlab, que nos permite evaluar diferentes modelos de clasificacion a la vez ajustando parámetros como puede ser el k-fold, este parámetro indica el número de muestras a seleccionar dentro de las muestras de entrenamiento para realizar una validación preliminar de estos, en nuestro caso este valor será igual a 5 como se puede observar en la figura 3.1.

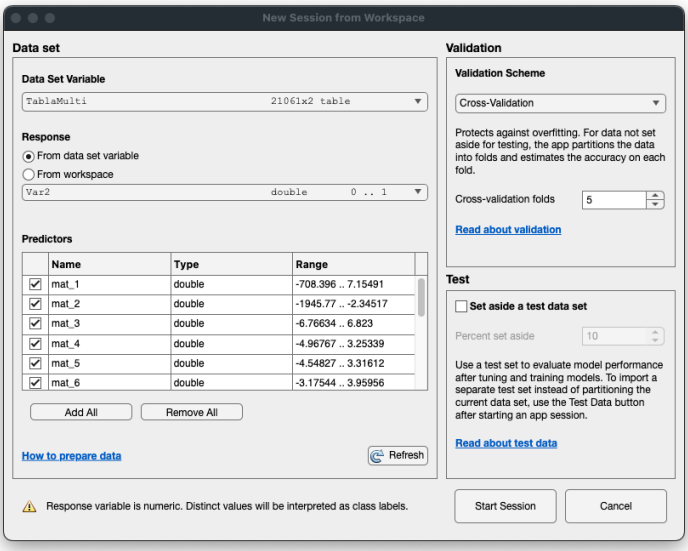

Figura 3.1 Introducción de datos de entrenamiento en el Classification Learner con k-fold igual a 5.

Una vez introducidos los datos procedemos a entrenar los clasificadores obteniendo los siguientes resultados en una ventana como la mostrada en la figura 3.2.

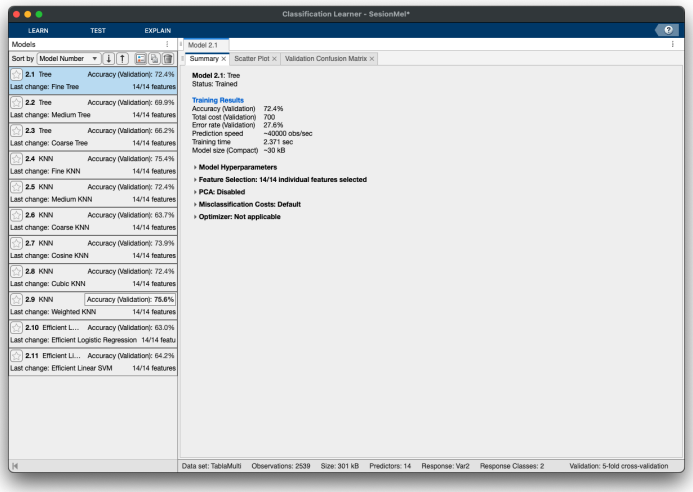

Figura 3.2 Resultado entrenamiento del Classification Learner.

Los resultados que hemos obtenido al realizar el entrenamiento se muestran en la figura 3.3 en la que se comparan la precisión de los distintos clasificadores al ser evaluados con el split del k-fold, el resultado para el entrenamiento utilizando embeddings está representado mediante las barras de color azul, mientras que las de los coeficientes de mel están representadas en color rojo.

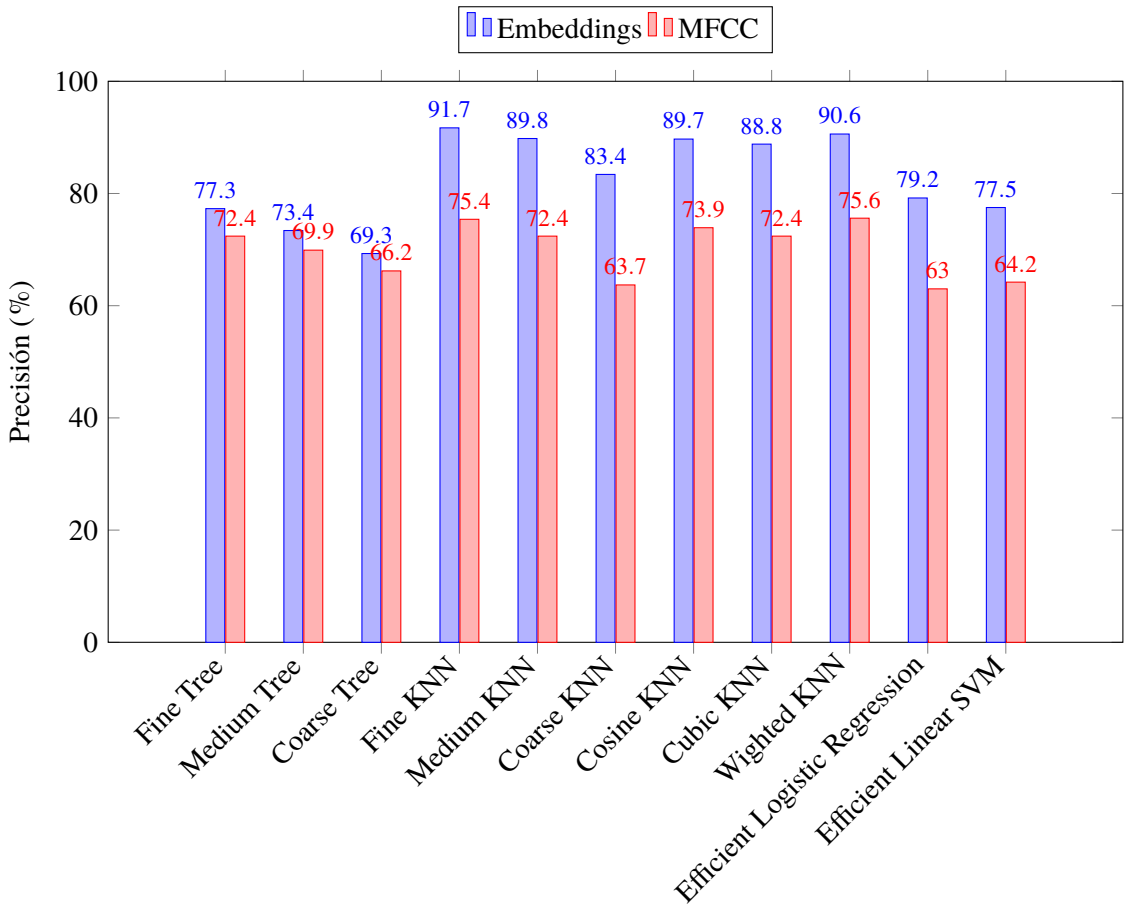

Figura 3.3 Comparativa de la precisión de evaluación en el entrenamiento por frames con k-fold=5.

A priori, comparando los tres clasificadores con mejores resultados de cada tipo de caracteristicas en las tablas 3.1, se puede observar que los resultados son más prometedores con los embeddings,.

Tabla 3.1 Tres mejores clasificadores durante la validación.

Método	Precisión Embeddings
Fine KNN	91,7 %
Wighted KNN	90,6 %
Medium KNN	89,8 %

Método	Precisión MFCC
Weighted KNN	75,6 %
Fine KNN	75,4 %
Cosine KNN	73,9 %

3.2 Resultados de la seleccion del umbral de clasificacion

Para evaluar los resultados de nuestro sistema es necesario realizar un último ajuste a los clasificadores que hemos entrenado, pues estos funcionan con frames individuales, sin embargo, cada muestra de sonido está compuesta de un número variable de frames, por lo que en esta fase calcularemos un umbral de detección a partir del cual una muestra se considerará perteneciente a una Alarma. Para ello, utilizando el split de evaluación completo de FSD50K, evaluaremos las ecuaciones de Exactitud (Accuracy), Precisión (Precision) y la Tasa de verdaderos positivos (Recall). La exactitud indica el porcentaje de veces que el sistema acierta en total, la precisión indica el porcentaje de veces que el sistema acierta clasificando la clase objetivo y la tasa de verdaderos positivos indica con qué porcentaje el sistema es capaz de detectar todos los eventos de la clase objetivo. Estas medidas vienen dadas por:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

en las que TP son los verdaderos positivos, TN son los verdaderos negativos, FP son los falsos positivos y FN son los falsos negativos.

Tras realizar la evaluación de los embeddings de las muestras completas con distintos valores del umbral obtuvimos los siguientes resultados con el clasificador basado en el modelo Fine KNN:

- Para el umbral de 0.3 se obtienen los siguientes resultados con embeddings:

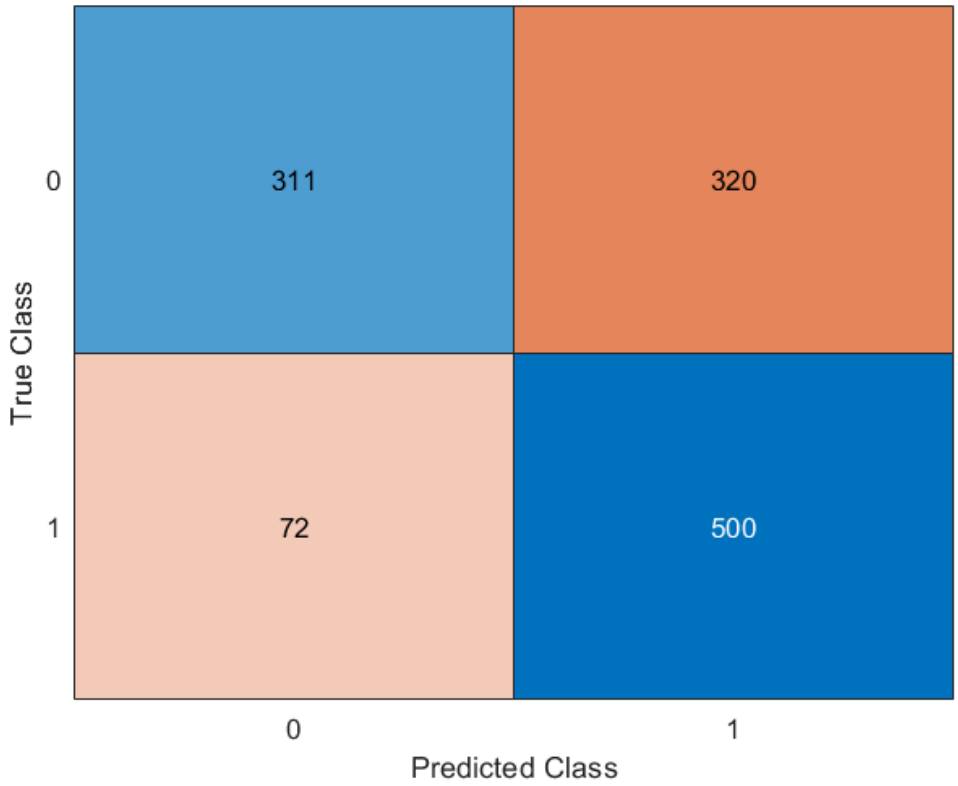

Figura 3.4 Matriz de confusión de embeddings con umbral 0.3.

$$\text{Accuracy} = \frac{500+311}{500+311+320+72} = \frac{811}{1203} = 0.674 \tag{3.4}$$

$$\text{Precision} = \frac{500}{500+320} = \frac{500}{820} = 0.61 \tag{3.5}$$

$$\text{Recall} = \frac{500}{500+72} = \frac{500}{572} = 0.874 \tag{3.6}$$

- Para el umbral de 0.4 se obtienen los siguientes resultados con embeddings:

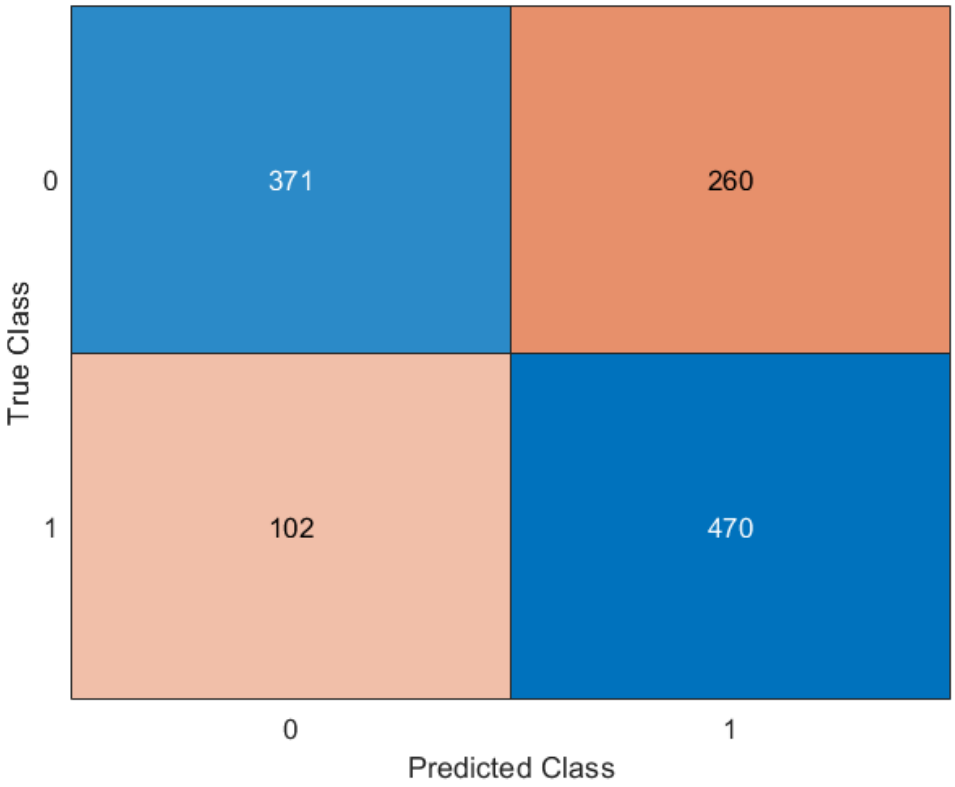

Figura 3.5 Matriz de confusión de embeddings con umbral 0.4.

$$\text{Accuracy} = \frac{470+371}{470+371+260+102} = \frac{841}{1203} = 0.699 \tag{3.7}$$

$$\text{Precision} = \frac{470}{470+260} = \frac{470}{730} = 0.644 \tag{3.8}$$

$$\text{Recall} = \frac{470}{470+102} = \frac{470}{572} = 0.822 \tag{3.9}$$

- Para un umbral de 0.5 se obtienen los siguientes resultados con embeddings:

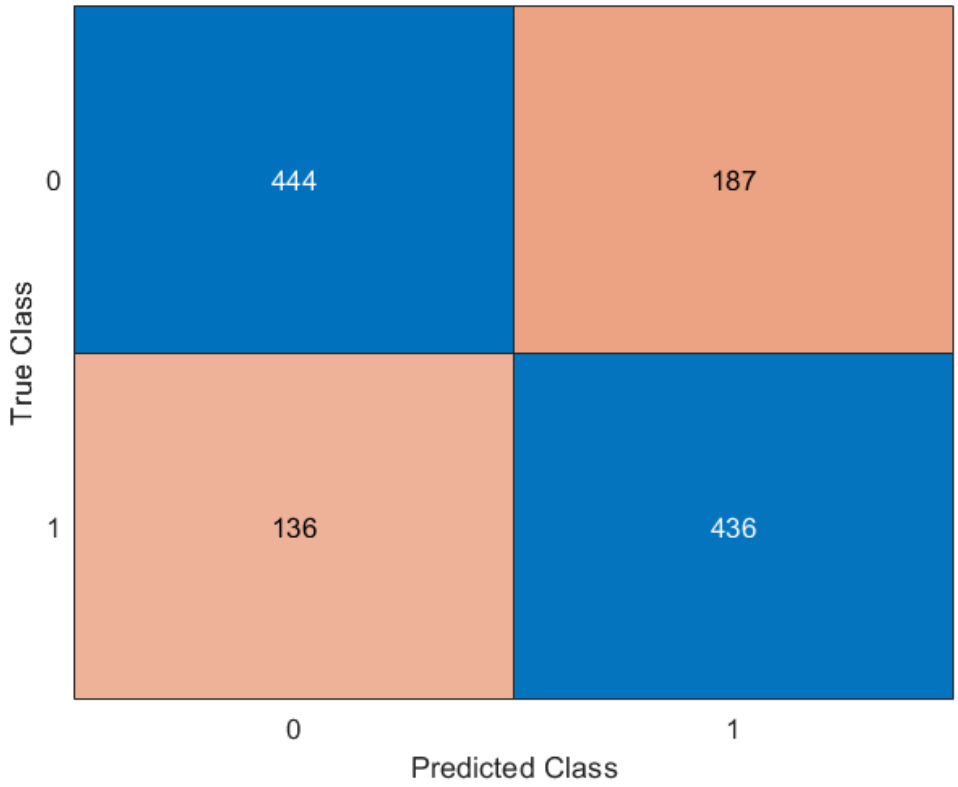

Figura 3.6 Matriz de confusión de embeddings con umbral 0.5.

$$\text{Accuracy} = \frac{436+444}{436+444+187+136} = \frac{880}{1203} = 0.732 \tag{3.10}$$

$$\text{Precision} = \frac{436}{436+187} = \frac{436}{623} = 0.7 \tag{3.11}$$

$$\text{Recall} = \frac{436}{436+136} = \frac{436}{572} = 0.762 \tag{3.12}$$

- Para el umbral de 0.6 se obtienen los siguientes resultados con embeddings:

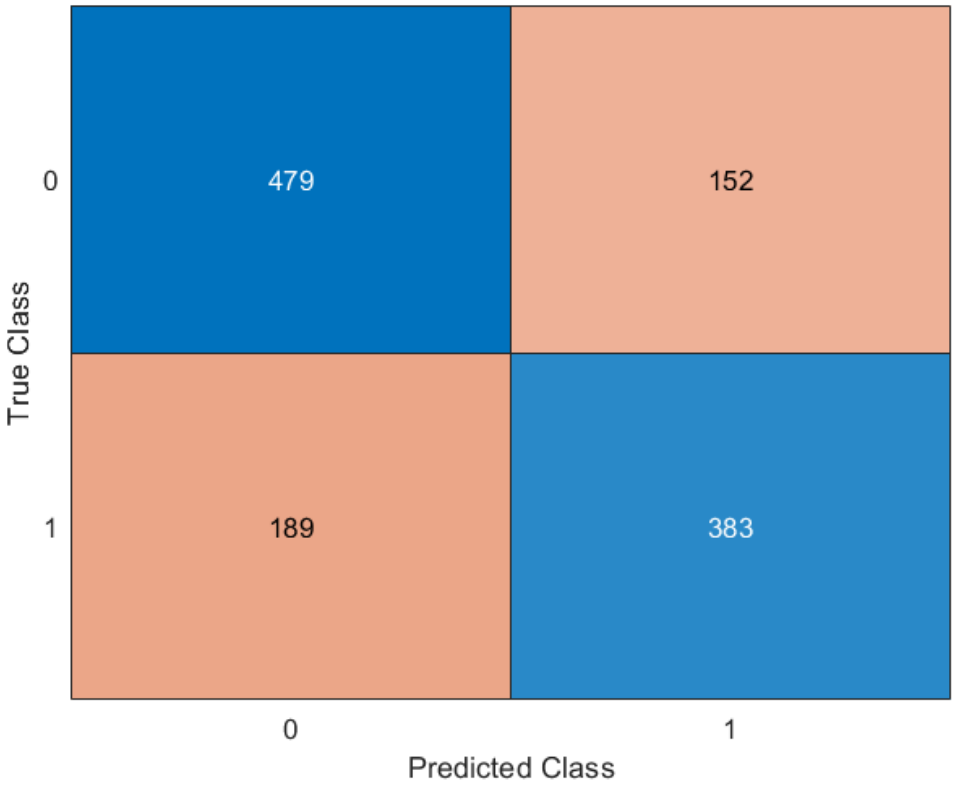

Figura 3.7 Matriz de confusión de embeddings con umbral 0.6.

$$\text{Accuracy} = \frac{383+479}{383+479+152+189} = \frac{862}{1203} = 0.717 \tag{3.13}$$

$$\text{Precision} = \frac{383}{383+152} = \frac{383}{535} = 0.716 \tag{3.14}$$

$$\text{Recall} = \frac{383}{383+189} = \frac{383}{572} = 0.7 \tag{3.15}$$

- Para el umbral de 0.7 se obtienen los siguientes resultados con embeddings:

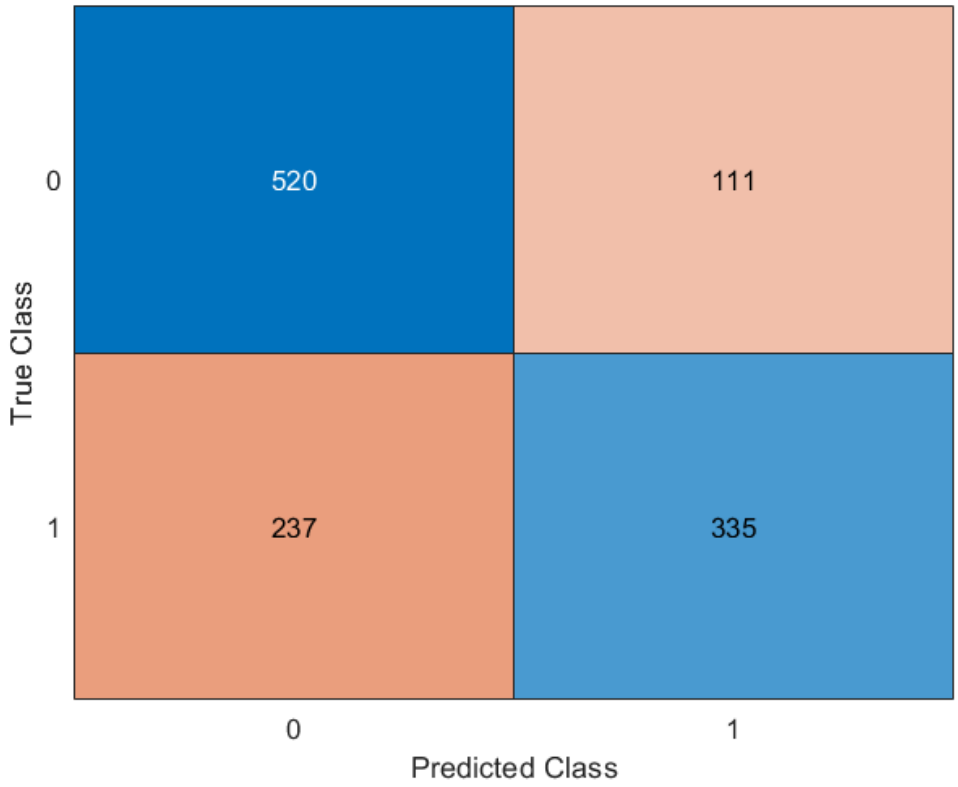

Figura 3.8 Matriz de confusión de embeddings con umbral 0.7.

$$\text{Accuracy} = \frac{335+520}{335+520+111+237} = \frac{855}{1203} = 0.711 \tag{3.16}$$

$$\text{Precision} = \frac{335}{335+111} = \frac{335}{446} = 0.751 \tag{3.17}$$

$$\text{Recall} = \frac{335}{335+237} = \frac{335}{572} = 0.586 \tag{3.18}$$

Tras realizar la evaluación de las características MFCC de las muestras completas con distintos valores del umbral obtuvimos los siguientes resultados con el clasificador basado en el modelo Weighted KNN: :

- Para el umbral de 0.3 se obtienen los siguientes resultados con coeficientes de Mel:

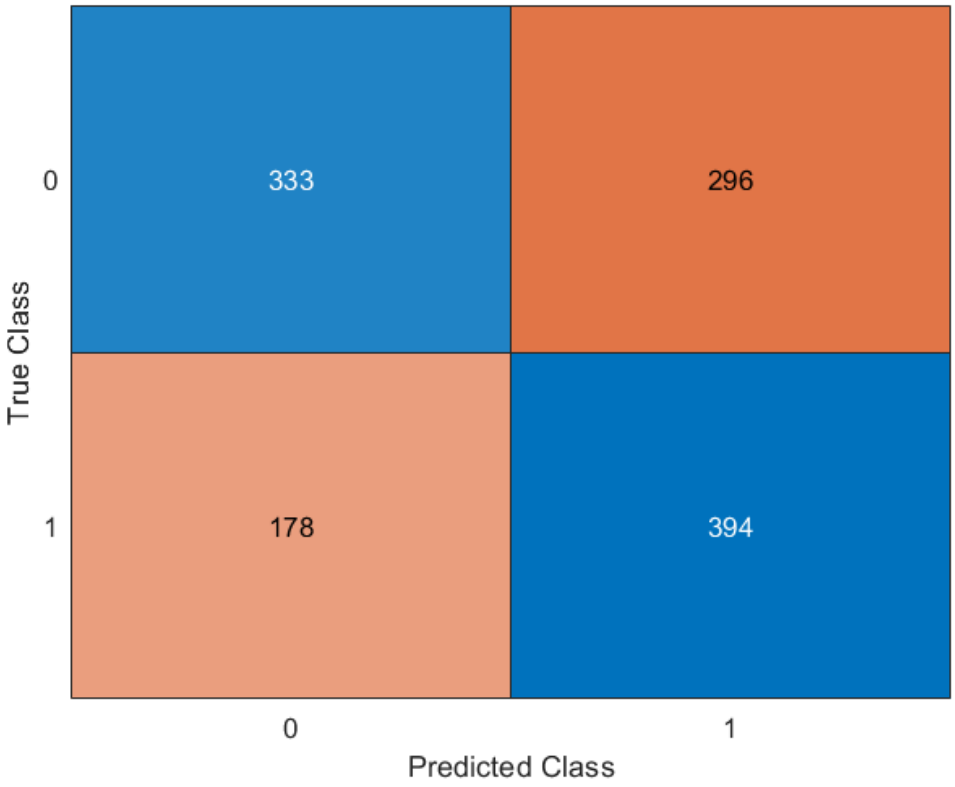

Figura 3.9 Matriz de confusión MFCC con umbral 0.3.

$$\text{Accuracy} = \frac{394+333}{394+333+296+178} = \frac{727}{1201} = 0.605 \tag{3.19}$$

$$\text{Precision} = \frac{394}{394+296} = \frac{394}{690} = 0.571 \tag{3.20}$$

$$\text{Recall} = \frac{394}{394+178} = \frac{394}{572} = 0.689 \tag{3.21}$$

- Para el umbral de 0.4 se obtienen los siguientes resultados con coeficientes de Mel:

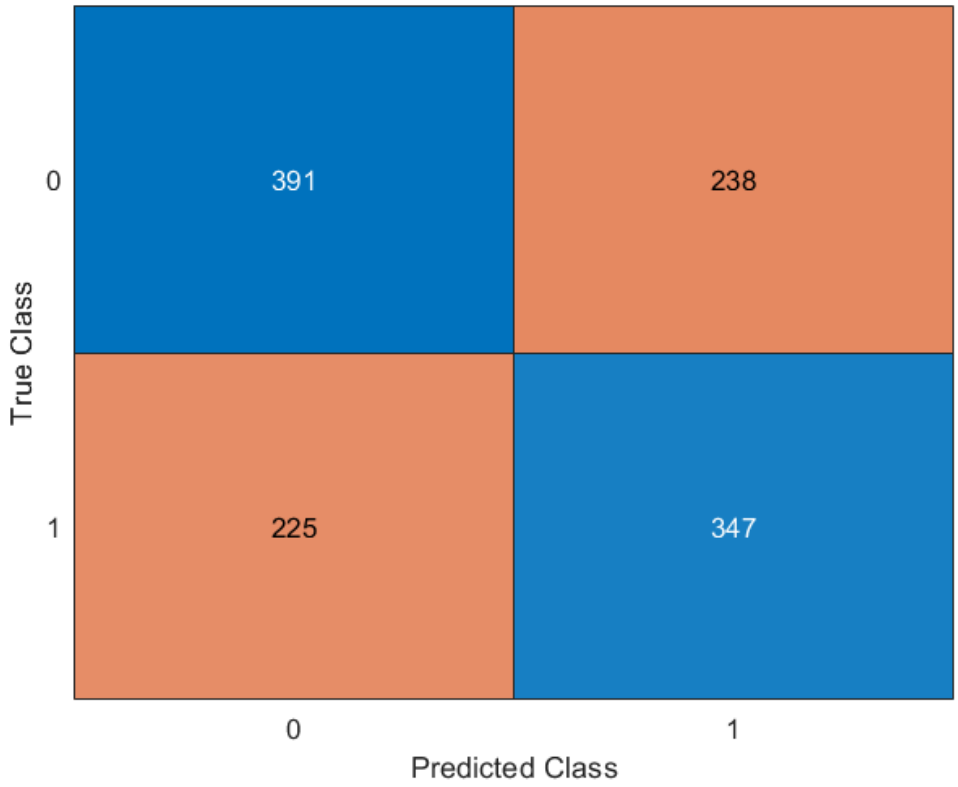

Figura 3.10 Matriz de confusión MFCC con umbral 0.4.

$$\text{Accuracy} = \frac{347+391}{347+391+238+225} = \frac{738}{1201} = 0.614 \tag{3.22}$$

$$\text{Precision} = \frac{347}{347+238} = \frac{347}{585} = 0.593 \tag{3.23}$$

$$\text{Recall} = \frac{347}{347+225} = \frac{347}{572} = 0.607 \tag{3.24}$$

- Para el umbral de 0.5 se obtienen los siguientes resultados con coeficientes de Mel:

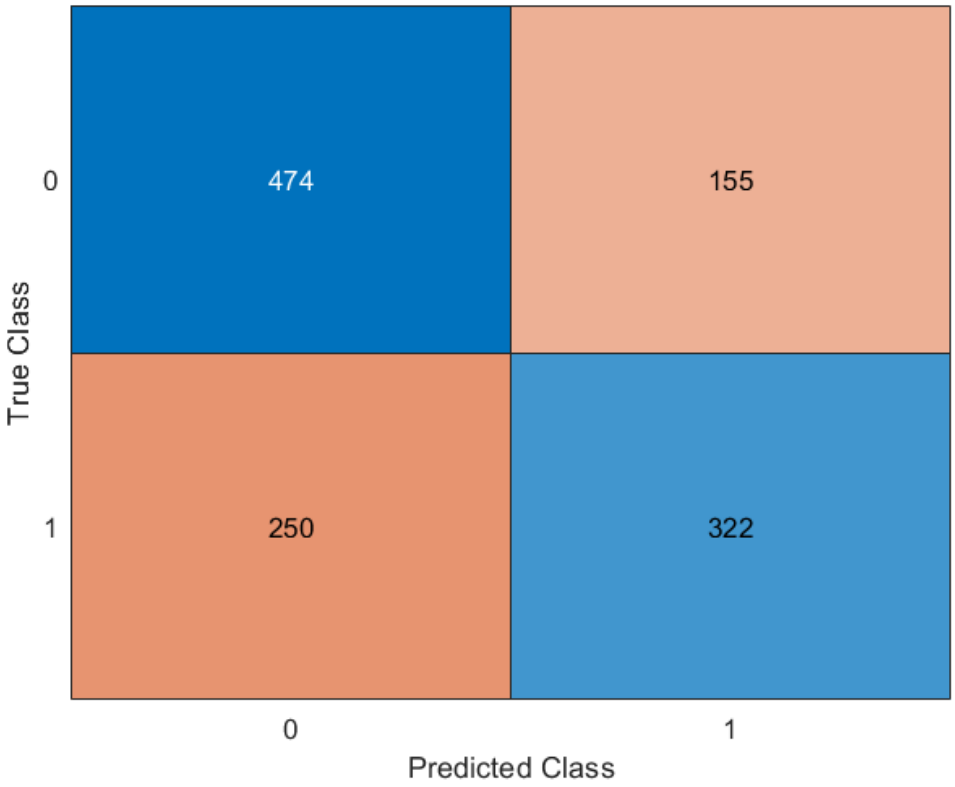

Figura 3.11 Matriz de confusión MFCC con umbral 0.5.

$$\text{Accuracy} = \frac{322+474}{322+474+155+250} = \frac{796}{1201} = 0.663 \tag{3.25}$$

$$\text{Precision} = \frac{322}{322+155} = \frac{322}{477} = 0.675 \tag{3.26}$$

$$\text{Recall} = \frac{322}{322+250} = \frac{322}{572} = 0.563 \tag{3.27}$$

- Para el umbral de 0.6 se obtienen los siguientes resultados con coeficientes de Mel:

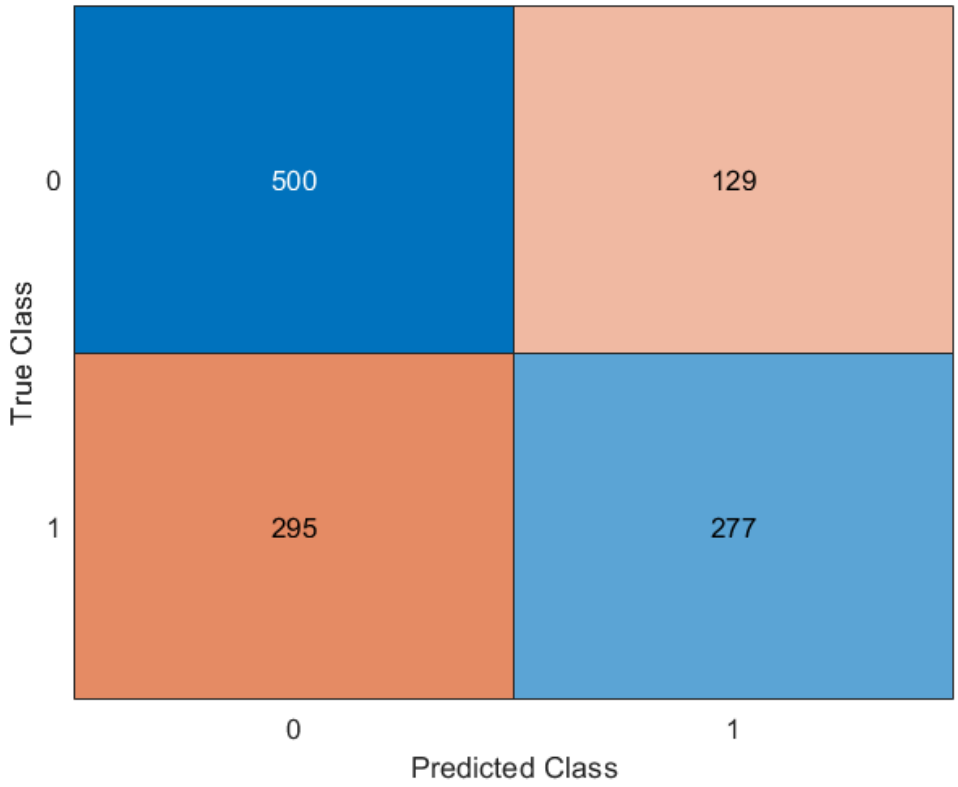

Figura 3.12 Matriz de confusión MFCC con umbral 0.6.

$$\text{Accuracy} = \frac{277+500}{277+500+129+295} = \frac{777}{1201} = 0.647 \tag{3.28}$$

$$\text{Precision} = \frac{277}{277+129} = \frac{277}{406} = 0.682 \tag{3.29}$$

$$\text{Recall} = \frac{277}{277+295} = \frac{277}{572} = 0.484 \tag{3.30}$$

- Para el umbral de 0.7 se obtienen los siguientes resultados con coeficientes de Mel:

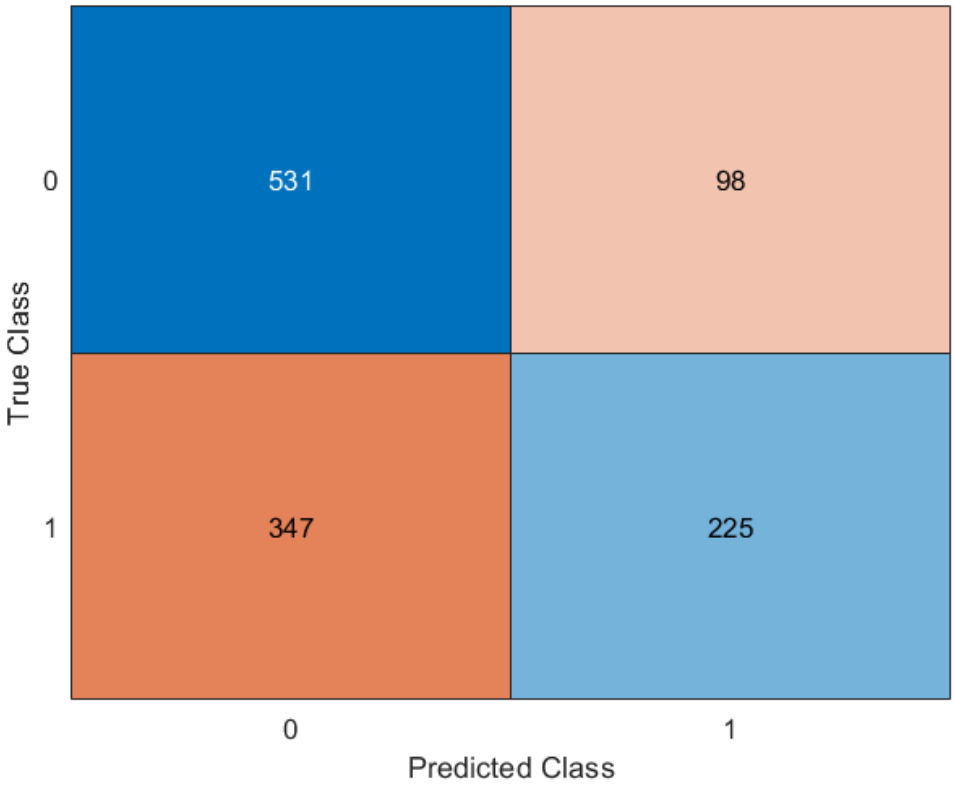

Figura 3.13 Matriz de confusión MFCC con umbral 0.7.

$$\text{Accuracy} = \frac{225+531}{225+531+98+347} = \frac{756}{1201} = 0.629 \tag{3.31}$$

$$\text{Precision} = \frac{225}{225+98} = \frac{225}{323} = 0.697 \tag{3.32}$$

$$\text{Recall} = \frac{225}{225+347} = \frac{225}{572} = 0.393 \tag{3.33}$$

Finalmente, con los resultados obtenidos utilizando diferentes umbrales para los embeddings en la tabla 3.2 y los resultados para las características MFCC en la tabla 3.3 podemos llegar a la conclusión de que el mejor umbral en ambos casos es el de 0,5, ya que es el que tiene las métricas más equilibradas en los tres parámetros que hemos evaluado. El resultado con este umbral es equivalente al del resultado final, pues hemos utilizado el split de evaluación al completo.

Tabla 3.2 Tabla comparativa de los resultados con embeddings según el umbral de decisión.

Umbral	Accuracy (Exactitud)	Precision (Precisión)	Recall (Tasa de verdaderos positivos)
0,3	67,4 %	61 %	87,4 %
0,4	69,9 %	64,4 %	82,2 %
0,5	73,2 %	70 %	76,2 %
0,6	71,7 %	71,6 %	70 %
0,7	71,1 %	75,1 %	58,6 %

Tabla 3.3 Tabla comparativa de los resultados con coeficientes de Mel según el umbral de decisión.

Umbral	Accuracy (Exactitud)	Precision (Precisión)	Recall (Tasa de verdaderos positivos)
0,3	60,5 %	57,1 %	68,9 %
0,4	61,4 %	59,3 %	60,7 %
0,5	66,3 %	67,5 %	56,3 %
0,6	64,7 %	68,2 %	48,4 %
0,7	62,9 %	69,7 %	39,3 %

3.3 Resultado final

Tras seleccionar el umbral de 0,5 para realizar la clasificación de las muestras hemos ejecutado del SED con el split completo de evaluación de FSD50K para calcular los parámetros finales del sistema con los siguientes resultados:

- Resultados obtenidos de clasificación con embeddings:

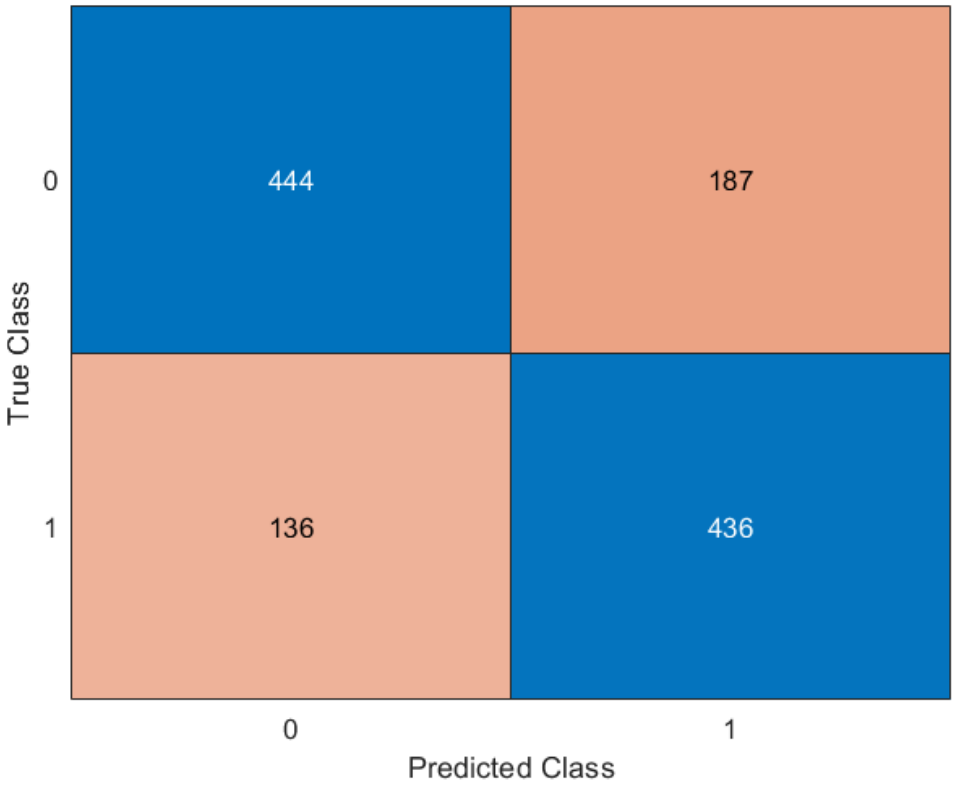

Figura 3.14 Matriz de confusión del split de evaluación con embeddings.

$$\text{Accuracy} = \frac{436+444}{438+444+187+136} = \frac{880}{1203} = 0.732 \tag{3.34}$$

$$\text{Precision} = \frac{436}{436+187} = \frac{436}{623} = 0.7 \tag{3.35}$$

$$\text{Recall} = \frac{436}{436+136} = \frac{436}{572} = 0.762 \tag{3.36}$$

- Resultados obtenidos de clasificación con características MFCC:

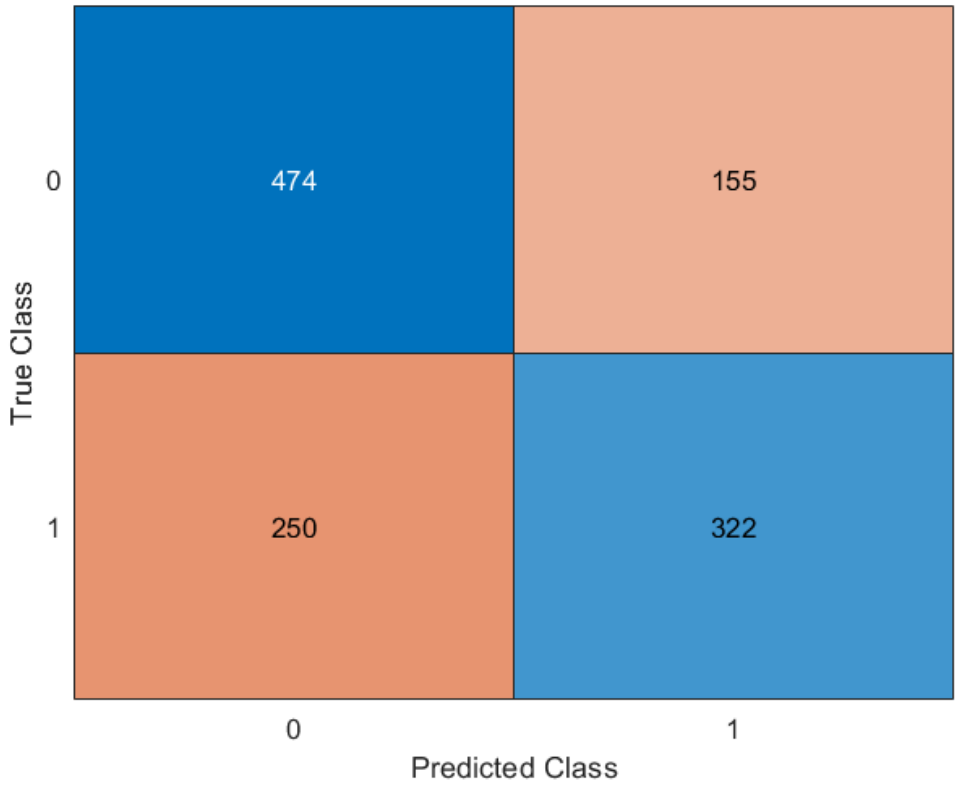

Figura 3.15 Matriz de confusión del split de evaluación con características MFCC.

$$\text{Accuracy} = \frac{322+474}{322+474+155+250} = \frac{796}{1201} = 0.663 \tag{3.37}$$

$$\text{Precision} = \frac{322}{322+155} = \frac{322}{477} = 0.675 \tag{3.38}$$

$$\text{Recall} = \frac{322}{322+250} = \frac{322}{572} = 0.563 \tag{3.39}$$

Los resultados finales se pueden observar con mas facilidad para su comparación en la tabla 3.4:

Tabla 3.4 Tabla comparativa de los resultados finales según el tipo de características.

Características	Accuracy (Exactitud)	Precision (Precisión)	Recall (Tasa de verdaderos positivos)
Embeddings	73,2 %	70 %	76,2 %
MFCC	66,3 %	67,5 %	56,3 %

3.4 Conclusiones

Una vez elegido el modelo entrenado la exactitud final del sistema estudiado ha sido de un 73,2 %, la precisión ha sido de un 70 % y su tasa de verdaderos positivos es de un 76,2 %, lo que nos permite afirmar que sería lo suficientemente robusto como para aplicarlo en un sistema con aplicaciones reales. Además, es un resultado superior al SED que utiliza características MFCC en todas las métricas, siendo el SED que utiliza embeddings un 7 % superior en exactitud, un 2,5 % en precisión y un 19,9 % superior en la tasa de verdaderos positivos. Aun así las dificultades encontradas con la base de datos en el etiquetado de las muestras, inducen a pensar que quizás los valores obtenidos en la evaluación propia del sistema de detección de eventos sonoros estén sesgados. Algunas de las muestras que hemos escuchado no correspondían con la etiqueta que tenían, además de que las inconsistencias de la jerarquía ha limitado mucho la profundidad de este estudio. Este proyecto también es muy intensivo computacionalmente por lo que ha habido que dedicar mucho tiempo simplemente al cálculo de modelos alternativos o posibles soluciones a los múltiples impedimentos que nos hemos encontrado, creando un cuello de botella importante en el proceso de desarrollo de este SED. Algunas de estas ideas serán expuestas más adelante. Pese a las limitaciones inherentes a el desarrollo de un proyecto como este en un tema que puede ser tan versátil y con tantas aplicaciones se ha llegado a un modelo funcional y se han identificado problemas y soluciones que pueden ser explotadas en un futuro a la hora de mejorar tanto la base de datos como las aplicaciones de MATLAB que hemos utilizado de apoyo en el diseño del sistema.

4 Líneas Futuras

El presente capítulo se dedica a la exploración de ideas y conceptos que, si bien no pudieron ser implementados durante el desarrollo actual del proyecto, representan vías para la mejora del Sistema de Detección de Eventos Sonoros (SED) en futuras iteraciones de la investigación. Es importante destacar que, aunque estas ideas no han sido implementadas en la presente iteración del proyecto, su discusión y análisis son fundamentales para la comprensión integral de las posibilidades y limitaciones actuales de los SED.

4.1 Base de datos

En el ámbito de la base de datos la mejora principal que se podría realizar es en el aspecto del etiquetado. Al tratarse FSD50K de una base de datos con etiquetado débil, una posible mejora de cara al futuro sería un refuerzo del etiquetado. Escuchando las muestras, corrigiendo errores de etiquetado, mejorando la jerarquía del etiquetado, añadiendo datos a las etiquetas de la posición del evento. Todo esto sin alterar el uso de la ontología de AudioSet.

4.2 Aumento del número de clases

Una vez realizado el proceso descrito anteriormente se podría aumentar la cantidad de clases que se podría añadir al clasificador. Actualmente solo tiene las Alarmas debido a la cantidad ingente de muestras que tiene la base de datos, una vez corregido el problema de las etiquetas se podría añadir más eventos al modelo para añadir versatilidad y usos potenciales al detector.

4.3 Portar el SED a Python

Pese a que MATLAB no deja de ser una herramienta excepcional, el auge de Python como lenguaje de programación y sus librerías de acceso abierto suponen una ventaja importante frente al uso de licencias de MATLAB, para hacer más accesible este trabajo y sus posibles mejoras sería conveniente crear una versión utilizando Python y sus librerías.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Annamaria Mesaros, Toni Heittola, Antti Eronen, and Tuomas Virtanen. Acoustic event detection in real life recordings. In *2010 18th European Signal Processing Conference*, pages 1267–1271, 2010.

[2] Axel Plinge, René Grzeszick, and Gernot A. Fink. A bag-of-features approach to acoustic event detection. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3704–3708, 2014.

[3] Pasquale Foggia, Nicolai Petkov, Alessia Saggese, Nicola Strisciuglio, and Mario Vento. Reliable detection of audio events in highly noisy environments. *Pattern Recognition Letters*, 65:22–28, 2015.

[4] Chan Teck Kai and Cheng Siong Chin. A comprehensive review of polyphonic sound event detection. *IEEE Access*, PP:1–1, 06 2020.

[5] Emmanouil Benetos, Grégoire Lafay, Mathieu Lagrange, and Mark Plumbley. Polyphonic sound event tracking using linear dynamical systems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25, 06 2017.

[6] JangBumsuk and LeeSang-Hyun. Cnn based sound event detection method using nmf preprocessing in background noise environment. *International journal of advanced smart convergence*, 9(2):20–27, 06 2020.

[7] Emre Cakir, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen. Polyphonic sound event detection using multi label deep neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2015.

[8] Karol J. Piczak. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2015.

[9] Giambattista Parascandolo, Heikki Huttunen, and Tuomas Virtanen. Recurrent neural networks for polyphonic sound event detection in real life recordings. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6440–6444, 2016.

[10] Emre Çakır, Giambattista Parascandolo, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen. Convolutional recurrent neural networks for polyphonic sound event detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(6):1291–1303, 2017.

[11] Ainuddin Wahid Abdul Wahab Shahaboddin Shamshirband Elham Babaee, Nor Badrul Anuar and Anthony T. Chronopoulos. An overview of audio event detection methods from feature extraction to classification. *Applied Artificial Intelligence*, 31(9-10):661–714, 2017.

[12] Theodoros Giannakopoulos and Aggelos Pikrakis. *Introduction to Audio Analysis: A MATLAB Approach*. Academic Press, Inc., USA, 1st edition, 2014.

[13] Zrar Kh. Abdul and Abdulbasit K. Al-Talabani. Mel frequency cepstral coefficient and its applications: A review. *IEEE Access*, 10:122136–122158, 2022.

[14] Xavier Valero and Francesc Alias. Gammatone cepstral coefficients: Biologically inspired features for non-speech audio classification. *IEEE Transactions on Multimedia*, 14(6):1684–1689, 2012.

[15] Jakob Abeßer, Sascha Grollmisch, and Meinard Müller. How robust are audio embeddings for polyphonic sound event tagging? *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:2658–2667, 2023.

[16] Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, R. Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron J. Weiss, and Kevin Wilson. Cnn architectures for large-scale audio classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 131–135, 2017.

[17] Anurag Kumar, Maksim Khadkevich, and Christian Fugen. Knowledge transfer from weakly labeled audio using convolutional neural network for sound events and scenes, 2018.

[18] Relja Arandjelović and Andrew Zisserman. Look, listen and learn, 2017.

[19] Aurora Linh Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. Look, listen, and learn more: Design choices for deep audio embeddings. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3852–3856, 2019.

[20] Sascha Grollmisch, Estefanía Cano, Christian Kehling, and Michael Taenzer. Analyzing the potential of pre-trained embeddings for audio classification tasks. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 790–794, 2021.

[21] Steve Young, G Evermann, M.J.F. Gales, Thomas Hain, D Kershaw, Xunying Liu, G Moore, James Odell, D Ollason, Daniel Povey, V Valtchev, and Philip Woodland. *The HTK book*. 01 2002.

[22] Douglas A Reynolds et al. Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663), 2009.

[23] Xiaodan Zhuang, Jing Huang, Gerasimos Potamianos, and Mark Hasegawa-Johnson. Acoustic fall detection using gaussian mixture models and gmm supervectors. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 69–72, 2009.

[24] Qing Zhou, Zuren Feng, and Emmanouil Benetos. Adaptive noise reduction for sound event detection using subband-weighted nmf. *Sensors*, 19(14), 2019.

[25] Dan Stowell, Dimitrios Giannoulis, Emmanouil Benetos, Mathieu Lagrange, and Mark Plumbley. Detection and classification of acoustic scenes and events. *IEEE Transactions on Multimedia*, 17:1733–1746, 10 2015.

[26] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. Tut database for acoustic scene classification and sound event detection. In *2016 24th European Signal Processing Conference (EUSIPCO)*, pages 1128–1132, 2016.

[27] Justin Salamon, Christopher Jacoby, and Juan Bello. A dataset and taxonomy for urban sound research. 11 2014.

[28] Karol J. Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, page 1015–1018, New York, NY, USA, 2015. Association for Computing Machinery.

[29] P Foster, S Sigtia, S Krstulovic, J Barker, and MD Plumbley. Chime-home: A dataset for sound source recognition in a domestic environment., 20151018 - 20151021.

[30] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 776–780, 2017.

[31] Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, and Xavier Serra. Fsd50k: An open dataset of human-labeled sound events. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:829–852, 2022.

[32] Yun-Ning Hung and Alexander Lerch. Feature-informed embedding space regularization for audio classification. In *2022 30th European Signal Processing Conference (EUSIPCO)*, pages 419–423, 2022.